

*Commenced Publication in 1973*

Founding and Former Series Editors:

Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

## Editorial Board

David Hutchison

*Lancaster University, UK*

Takeo Kanade

*Carnegie Mellon University, Pittsburgh, PA, USA*

Josef Kittler

*University of Surrey, Guildford, UK*

Jon M. Kleinberg

*Cornell University, Ithaca, NY, USA*

Alfred Kobsa

*University of California, Irvine, CA, USA*

Friedemann Mattern

*ETH Zurich, Switzerland*

John C. Mitchell

*Stanford University, CA, USA*

Moni Naor

*Weizmann Institute of Science, Rehovot, Israel*

Oscar Nierstrasz

*University of Bern, Switzerland*

C. Pandu Rangan

*Indian Institute of Technology, Madras, India*

Bernhard Steffen

*TU Dortmund University, Germany*

Madhu Sudan

*Microsoft Research, Cambridge, MA, USA*

Demetri Terzopoulos

*University of California, Los Angeles, CA, USA*

Doug Tygar

*University of California, Berkeley, CA, USA*

Gerhard Weikum

*Max-Planck Institute of Computer Science, Saarbruecken, Germany*

Stefano Ceri Marco Brambilla (Eds.)

# Search Computing

Challenges and Directions

Volume Editors

Stefano Ceri  
Marco Brambilla  
Politecnico di Milano  
Dipartimento di Elettronica e Informazione  
Piazza L. da Vinci, 32, 20133 Milano, Italy  
E-mail: {ceri,mbrambil}@elet.polimi.it

Library of Congress Control Number: 2010923485

CR Subject Classification (1998): H.4, H.3, D.4, C.2.4, F.2, D.1.3

LNCS Sublibrary: SL 3 – Information Systems and Applications, incl. Internet/Web and HCI

ISSN 0302-9743  
ISBN-10 3-642-12309-0 Springer Berlin Heidelberg New York  
ISBN-13 978-3-642-12309-2 Springer Berlin Heidelberg New York

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, re-use of illustrations, recitation, broadcasting, reproduction on microfilms or in any other way, and storage in data banks. Duplication of this publication or parts thereof is permitted only under the provisions of the German Copyright Law of September 9, 1965, in its current version, and permission for use must always be obtained from Springer. Violations are liable to prosecution under the German Copyright Law.

springer.com

© Springer-Verlag Berlin Heidelberg 2010  
Printed in Germany

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India  
Printed on acid-free paper 06/3180

## Preface

Who are the strongest European competitors on software ideas? Who is the best doctor to cure insomnia in a nearby hospital? Where can I attend an interesting conference in my field close to a sunny beach? This information is available on the Web, but no software system can accept such queries nor compute the answer. At most, users can identify sub-problems that can be addressed by specific search engines and interact with each of them serially, but then they have the responsibility of building global answers by manually composing results. Search computing is a new multi-disciplinary discipline which will provide the abstractions, methods, tools and computing systems required to express these queries and to build their answer.

The emerging paradigm of software services has so far been neutral to search. Search computing is an evolution of service computing focused on building the answers of complex queries by interacting with a constellation of cooperating search services, using ranking as the dominant factor for service composition. New language and description paradigms are required for interconnecting services and for expressing queries. Semantic domain knowledge helps enrich terminological knowledge about objects being searched. New protocols help capture ranking preferences and their refinement; new interfaces present complex results with simple visual descriptions. Ranking is relative to individuals and context and therefore reflects personal and social contributions. Financial and legal implications of search computing must be understood and mastered. In summary, search computing is a multi-disciplinary effort which requires adding to sound software principles contributions from other sciences such as knowledge representation, human-computer interfaces, psychology, sociology, economics and legal sciences.

The Search Computing (Seco) Project is funded by the European Research Council (ERC), responding to the 2008 Call for “IDEAS Advanced Grants,” a program dedicated to the support of investigation-driven frontier research. SeCo started on November 1, 2008 and will last until October 31, 2013 (see [www.searchcomputing.eu](http://www.searchcomputing.eu).) This book describes the outcome of the first SeCo “Workshop on Search Computing Challenges and Directions,” held in Como during June 17–19, 2009.

The book is divided into three parts. The first part presents visions of the current evolution in search, which is becoming more and more task-oriented and is now starting to use ontological knowledge in order to manage complex queries; these visions are marking the new trends in search.

The second part provides some *background and related technologies*. These can be considered as parallel fields of research, useful both for setting the theoretical premises for search computing and for providing a technological framework for building search computing systems and applications.



The third part dwells on the *technological problems and issues* which arise when dealing with search computing as a new search paradigm. It provides a unified view of the results of search computing as achieved exactly one year after its starting date.

The book is the result of a collective effort of all the project participants and has been reviewed with the help of the project's advisory board members and of several other experts. We thank all of them for their effort.

January 2010

Stefano Ceri  
Marco Brambilla

# Organization

## Reviewers

Luciano Baresi	Politecnico di Milano
Marco Brambilla	Politecnico di Milano
Fabio Casati	Università di Trento
Tiziana Catarci	Università di Roma “La sapienza”
Stefano Ceri	Politecnico di Milano
Georg Gottlob	Oxford University
Ihab Ilyas	University of Waterloo
Ioana Manolescu	INRIA and Université de Paris Sud
Giansalvatore Mecca	Università della Basilicata
Roberto Verganti	Politecnico di Milano
Gerhard Weikum	Max-Planck Institute for Informatics

## Sponsoring Institutions

The Search Computing (Seco) Project is funded by the European Research Council (ERC), responding to the 2008 Call for “IDEAS Advanced Grants”, a program dedicated to the support of investigation-driven frontier research. SeCo started on November 1st, 2008 and will last until October 31, 2013.

# Table of Contents

## Part I: Visions

Chapter 1: Search Computing . . . . .	3
<i>Stefano Ceri</i>	
Chapter 2: Next Generation Web Search . . . . .	11
<i>Ricardo Baeza-Yates and Prabhakar Raghavan</i>	
Chapter 3: Search for Knowledge . . . . .	24
<i>Gerhard Weikum</i>	

## Part II: Technology Watch for Search Computing

Chapter 4: The Search Engine Industry . . . . .	45
<i>Tommaso Buganza and Emanuele Della Valle</i>	
Chapter 5: From Mashup Technologies to Universal Integration: Search Computing the Imperative Way . . . . .	72
<i>Florian Daniel, Stefano Soi, and Fabio Casati</i>	
Chapter 6: Web Data Extraction for Service Creation . . . . .	94
<i>Robert Baumgartner, Alessandro Campi, Georg Gottlob, and Marcus Herzog</i>	
Chapter 7: Dataspaces . . . . .	114
<i>Cornelia Hedeler, Khalid Belhajjame, Norman W. Paton, Alessandro Campi, Alvaro A.A. Fernandes, and Suzanne M. Embury</i>	
Chapter 8: Multimedia and Multimodal Information Retrieval . . . . .	135
<i>Alessandro Bozzon and Piero Fraternali</i>	

## Part III: Issues in Search Computing

Chapter 9: Service Marts . . . . .	163
<i>Alessandro Campi, Stefano Ceri, Georg Gottlob, Andrea Maesani, and Stefania Ronchi</i>	
Chapter 10: Join Methods and Query Optimization . . . . .	188
<i>Daniele Braga, Stefano Ceri, and Michael Grossniklaus</i>	
Chapter 11: Rank-Join Algorithms for Search Computing . . . . .	211
<i>Ihab F. Ilyas, Davide Martinenghi, and Marco Tagliasacchi</i>	

Chapter 12: Panta Rhei: Flexible Execution Engine for Search Computing Queries .....	225
<i>Daniele Braga, Stefano Ceri, Francesco Corcoglioniti, and Michael Grossniklaus</i>	
Chapter 13: Liquid Queries and Liquid Results in Search Computing ...	244
<i>Alessandro Bozzon, Marco Brambilla, Stefano Ceri, Piero Fraternali, and Ioana Manolescu</i>	
Chapter 14: Building Search Computing Applications .....	268
<i>Alessandro Bozzon, Marco Brambilla, Stefano Ceri, Francesco Corcoglioniti, and Nicola Gatti</i>	
Chapter 15: Search Computing and the Life Sciences .....	291
<i>Marco Masseroli, Norman W. Paton, and Irena Spasić</i>	
<b>Appendixes</b>	
Appendix A: Search Computing Dictionary .....	307
<b>Author Index</b> .....	321

**Part I**  
**Visions**

# Chapter 1: Search Computing

Stefano Ceri

Politecnico di Milano, Dipartimento di Elettronica ed Informazione,  
V. Ponzio 34/5, 20133 Milano, Italy  
stefano.ceri@polimi.it

**Abstract.** Search Computing is a new paradigm for composing search services. While state-of-art search systems answer generic or domain-specific queries, Search Computing enables answering questions via a constellation of dynamically selected, cooperating search services, which are correlated by means of join operations. The idea is simple, yet pervasive. New language and description paradigms are required for expressing queries and for connecting services. New user interfaces and protocols help capturing ranking preferences and enabling their refinement.

**Keywords:** Complex queries, multi-dimensional queries, search services, join operation, data integration, data visualization, process composition.

## 1 Beyond Page Search

Throughout the last decade, Internet search has been primarily performed by routing users towards the specific Web page that best answered their information needs. Major search engines, such as *Google*, *Yahoo* and *Bing*, crawl the Web and index Web pages, highlighting worldwide candidate “best” pages with excellent precision and recall; such ability has proven adequate to fulfill users’ needs, to the point that Web search is customarily performed by millions of users, both for work and leisure.

However, not all information needs can be satisfied by individual pages on the surface Web. On one hand, the so-called “deep Web” contains information which is perhaps more valuable than what can be crawled on the surface Web; on another side, as the users get confident in the use of search engines, their queries become more and more complex, to the point that their formulation goes beyond what can be expressed with a few keywords, their answers require more than a list of Web pages, and general-purpose search engines perform poorly upon them. According to search company’s experts, the number of complex queries that are not answered well by major search engines due to their intrinsic complexity is remarkably high and increasing. Many search interactions can be considered as part of a more complex process of expressing goals and achieving tasks, as discussed in the vision paper by Ricardo Baeza Yates (Chapter 2).

When a query addresses a specific domain (e.g., travels, music, shows, food, movies, health, and genetic diseases), domain-specific search engines do a better job

than general-purpose ones; but their expertise is focused upon a given domain. Thus, one can separately find best travel offers and interesting music shows, or conduct genetic analysis and investigate the related medical literature, but can hardly combine information from diverse yet related domains. An expert user can perform several independent searches and then manually combine the findings, but such procedure is cumbersome and error prone.

Search Computing aims at responding to **multi-domain queries**, i.e., queries over multiple semantic fields of interest, by helping users (or by substituting to them) in their ability to decompose queries and manually assemble complete results from partial answers; thus, Search Computing aims at filling the gap between generalized search systems, which are unable to find information spanning multiple topics, and domain-specific search systems, which cannot go beyond their domain limits.

Paradigmatic examples of Search Computing queries are: “Where can I attend an interesting scientific conference in my field and at the same time relax on a beautiful beach nearby?”, “Where is the theatre closest to my hotel, offering a high rank action movie and a near-by pizzeria?”, “Who are the strongest candidates in Europe for competing on software ideas?”, “Who is the best doctor who can cure insomnia in a nearby public hospital?”, “Which are the highest risk factors associated with the most prevalent diseases among the young population?” These examples show that Search Computing aims at covering a large and increasing spectrum of user’s queries, which structurally go beyond the capabilities of general-purpose search engines. These queries cannot be answered without capturing some of their semantics, which at minimum consists in understanding their underlying domains, in routing appropriate query subsets to each domain specific source and in combining answers from each expert to build a complete answer that is meaningful for the user.

## 2 State of the Art

Processing queries on multiple search engines is not new; **meta-search engines** are capable of routing the same query to multiple search engines and then presenting composite results. *Kosmix* is a new-generation meta-search engine connecting to over a thousand of sources by using their Web services. In *Kosmix*, the relevant data sources for a query are determined by matching the user’s keywords with a huge private concept taxonomy (of about a million nodes), after manually tagging the data sources with the same taxonomy concepts. *Kosmix*, then, routes the query to all data sources, without attempting source integration. Results are collected from *Google*, *Yahoo*, *Flicker*, *YouTube*, *Twitter*, and so on, and presented to users; sources typically include very popular search engines, but sometime also domain-specific sources, such as the *Day-Life* or *Slate* (in the news domain). While *Kosmix* has the ability of showing individual results from many distinct data sources, it doesn’t integrate multiple domains, and therefore cannot answer complex queries; rather, it can answer simple queries by retrieving data from a plurality of sources. Yet, *Kosmix* demonstrates that Web services are viable methods for getting information from remote sources.

**Vertical search engines** are focused upon a single domain, e.g., hotels (*Booking*) or flights (*Tuifly*), which are well-understood in terms of data quality and ranking

criteria (e.g., for hotels and flights ranking depends on price, plus other domain specific criteria, such as the hotel's location and stars, or the flight's duration and number of intermediate stops). Therefore, vertical search engine systems perform the ranking of search results by using a single scoring function and compute "global best" results for their domain. Compared to *Kosmix*, these systems are focused upon one single domain of expertise, but they compute "global" rankings and then order their results according to such ranking; instead, *Kosmix* collates results according to data source relevance, without intermixing items from the various sources. Also vertical search engines use Web services for connecting to data sources, although the number of data services available to a given engine is normally small.

Going beyond meta-search and vertical engines, we find **extensions of vertical search engines** capable of integrating information from multiple, but "contiguous", domains. For instance, *Expedia* or *Lastminute* are capable of integrating information about flights, hotels, car rentals, special events offerings, and so on. The fact that a user selects a flight with given associated departure/arrival times helps the system in proposing the proper length of the hotel stay or of the car rental, thus checking availability and prices and presenting a "best offer". Users normally are well aware of their travelling needs, therefore they select the trip first, and then acquire additional services; however, if they are offered a particularly attractive hotel booking, they can fix that choice and go back to flights, trying to improve their travel plan. Thus, an expert user can combine several travel services and work with combinations, improving each offer separately while maintaining them "connected" to the travel plan, and thus achieving an optimal "global offer". The notion of combination, based upon given destinations, dates and times, is very similar to the notion of composition that we want to develop in Search Computing; in a sense, *Expedia* and *Lastminute* are examples of Search Computing systems, however with given fixed domains and composition patterns.

Advancing search by using knowledge is raising a lot of interest, and is the topic of the vision paper by Gerhard Weikum (Chapter 3). **Knowledge-based search systems**, such as *Yago*, *Wolfram-Alpha* and *True Knowledge*, work by first building large ontologies and then translating user's queries into requests over such ontologies, thereby selecting the knowledge relevant to the answer. This method is certainly superior to conventional search for answering queries over well-structured and organized knowledge, e.g., Wikipedia (examples of such queries are Napoleon's year of birth, or city's populations and weather conditions, or the height of mountains in California).

Ontological search deeply differs from conventional search in that the work of crawlers is substituted by human-driven knowledge compilation; this is at the same time a virtue and a limitation of the method, as it cannot easily monitor evolving data - ontology evolution requires expert work, which currently is provided by humans at limited speed, and will hardly be capable of processing data about, e.g., daily events occurring worldwide. From our vision's perspective, ontology-based search are a new class of search systems, whose expertise is confined within a specific ontological description; they can be considered wider domain-specific systems, but they cannot exhaust the scope of complex search. However, these systems can overcome conventional search engines in their ability of providing answers whose content goes beyond the scope of an individual Web page.



### 3 Building Search Computing Systems

The essence of complex queries is their ability of extracting answers from complex data, rather than from within a single Web page; but complex data require a **data integration process**. Then, the fundamental question is whether such data integration process can be performed independently – and a-priori – from queries, or should instead be query-specific. In our vision, integration should be query-specific, since answering queries about travels and food or about genes and medical knowledge require intrinsically different data sources: building results for such queries does not require “global data integration”, but just data integration relative to specific domains. However, data integration is one of the hardest problems in computing, because it requires full understanding of the semantics of data sources; as such, it cannot be done without human intervention.

With Search Computing, we rely on the work of human experts too, but we move from ontology creation and management, a huge task, to **data source coupling**, a somewhat more feasible accomplishment. We denote as data source any data collection accessible on the Web. The Search Computing motto is that each data source should be focused on its single domain of expertise (e.g., travels, music, shows, food, movies, health, genetic diseases), but pairs of data sources which share information (e.g., about locations, people, genes) can be linked to each other, and then complex queries spanning over more than one data source can use such pairing (that we call “composition pattern”) to build complex results. An advantage of this approach is its transitivity: if we can pair source A to source B (e.g. pathologies which alter body functions), and then source B to source C (e.g. body functions alterations which are treated by drugs), then we can answer queries that connect A to C (e.g. pathologies treated by drugs) and so on. Each source is responsible of monitoring changes within its domain of expertise, e.g., movie offerings or airfares, through distributed and real-time processing that cannot be performed by knowledge managers, but should remain responsibility of the specialized data sources.

Then, the next problem to solve is how to build a **composition pattern**, i.e., a data source coupling for answering multi-domain queries, recalling that the purpose of composition is search, and that therefore results should be presented to users according to some ranking, respectful of the original rank of the elements coming from the native data sources and of the search intent of the user; indeed, users normally only look at top results of a search, therefore the composition pattern should enable a Search Computing system to produce the highest ranked results first. Our solution is to resort to join, the most popular data management operation, which is however revisited in the context of Search Computing to become service-based and ranking-aware. A result item of a multi-domain query is a “combination”, built by joining two or more elements coming from distinct data sources and returned by different search engines; in our first query example (“Where can I attend an interesting scientific conference in my field and at the same time relax on a beautiful beach nearby?”), combinations are triples made of: database conferences (extracted from a site specialized in scholar events, e.g., *Dblife*), inexpensive flights (extracted from a flight selection site, e.g., *Expedia* or *Edreams*), and cities with nice beaches (extracted from tourism or review sites, e.g., *Yahoo! Travel* or *Tripadvisor*). Connections carry semantics: flights connect pairs of cities at given dates; therefore

connections use “dates” and “cities” as matching properties. We apply joins to the context of software services, by assuming that every data source is wrapped as a web service, and that such services, in most cases, expose a query-like interface which assumes keyword-based input and produces ranked results as output. Services are then composed by using a ranking-preserving join. We regard such operation as a **join of search services**.

Search Computing aims at giving to expert users the capability of building similar solutions for different choices of domains, which – in the same way as *Expedia* or *Lastminute* – share given properties and therefore can be connected. For such purpose, Search Computing offers a collection of methods and techniques for orchestrating the search engines and building global results. Composition patterns are predefined connections between well-identified Web services, therefore orchestrations are not built arbitrarily, but rather by selecting nodes (representing services) and arcs (representing the links in the composition patterns) within a **resource network** representing the various knowledge sources and their connections. This vision is consistent with the emerging idea of moving from an Internet of (disorganized) pages to an Internet of (semantically coherent) objects.

With Search Computing, sources must be registered, and their composition patterns be established. This work requires human intervention, because sources can be linked only by means of join attributes, which must be type-compatible and describe the same real-world concept. **Source registration** occurs by describing the source properties and annotating their role (i.e., representing both input keyword and output result types); when two sources can be joined, a composition pattern is created and associated with a semantic description. This process builds a resource network; we envision communities of users sharing resources in large networks, but also private bodies (e.g., enterprises) developing their own proprietary network of related resources.

Then, query processing will use a **search computing framework**, consisting of a query optimizer – to decide the best order of execution of service calls and the best strategy for joining their results – and an execution engine – monitoring the progressive construction of results and achieving an optimal performance by means of producer-consumer paradigms implementing policies for balancing the frequency of calls to the various services, as well as various levels of caching. The execution engine supports joins of search services as the most relevant operation, and is equipped with mechanisms for regulating join speed to the pace of data production services. We foresee supporting the framework upon general-purpose distributed architectures, such as computing clouds, so as to be easily and effectively available to application providers.

Complex queries are not only hard to answer, but they can be also difficult to formulate for the user. There, an important stream of work is about capturing the user’s search intent and directing the user towards the discovery of his true information need. This process can be done by means of **liquid queries**, a dynamic query interface that lets users dynamically extend the scope of queries and then browse query results. We expect users to look at results both selectively and globally, possibly asking for more results from given sources, possibly performing grouping and aggregation operations upon result attributes, and so on. The design of the liquid query interface is inspired by *Google Squared*, whose concept is however extended by the fact that each portion of the result can be traced back to a well-defined data

source, thus offering the notion of “data provenance” within complex results. We use a variety of data visualization methods which highlight multiple dimensions and multiple rankings.

The link topology of the resource network also suggests a way to explore the information space by **augmenting query results**. Complex queries often imply that users have in mind a complex information finding task, which is better represented by an exploration process rather than by a one shot query; the resource network offers a natural way to expand the initial query or its results, by accessing nodes which are reachable from the nodes already used by a query (e.g., expanding the results about action movies by looking at additional information such as its director, actors, and so on, or expanding the notion of geo-localized theatres by looking at public transports or at the nearby pizzerias). Similar capabilities are offered by the latest releases of search systems, such as *Bing*, which however restricts query and result expansion to domains selected a priori; instead, the resource network could offer query-specific choices.

Finally, we consider the possibility of automatically inferring the relevant network of data sources required to build the answer from **keyword-based user queries**. This will require “understanding” query terms and associating them to resources, through tagging, matching, and clustering techniques; then, the query will be associated to the “best” network of resources according to matching functions, and dynamically evaluated upon them. This goal is rather ambitious, but it is similar to supporting automatic matching of query terms to services within a semantic network of concepts, currently offered by *Kosmix*. One step in this direction, that we are already considering, is to extend join between services to support the notions of partial linguistic matching between terms (supported by vocabularies such as WordNet) or dealing with the predicate “near” in specific domains (e.g. distance, time, money).

## 4 Building Search Computing Applications

We propose Search Computing as a new method for building a class of rank-aware information finding solutions, accessible to a vast community of Web application developers - and not necessarily confined to large search engine companies. This vision requires a vast community of data providers, who should instrument their data sources so as to become part of broader search environment. Therefore, we are concerned with finding a system of incentives so as to motivate the creation of communities of data providers and of application developers.

The trend towards supporting users in **publishing data sources on the Web** is a general one. Google, Yahoo and Microsoft are building environments and tools (*Fusion Tables, Yahoo! BOSS, Symphony*) for helping Web users to publish their data, with the goal of capturing the so-called “long tail” of data sources. We also consider data publishing essential for Search Computing, however with a specific connotation. Data sources should produce ranked output, organized as lists of items, so that data extraction can be performed incrementally, by “chunks” (sublists of a given number of results, e.g., 10 items fitting into a page), and users can suspend a search and then resume it, possibly guiding the way in which data sources should be inspected. This data organization, that we call “ranked and chunked”, is typically offered by search

service APIs (because answering a query normally requires the top few items), but it is not made available by most data sources. However, most data sources can be turned into “ranked and chunked”. Ranked data extraction is currently supported by query languages, and chunking can also be programmed on top of tabular data representations, by using top-k extraction commands. Such provisions apply to data which are initially materialized and mapped into suitable formats. Therefore, we are building tools and/or providing best practices, applicable to data sources of various kinds, for enabling data providers to build “search” service adapters. We design methods and tools which will take into account the most popular data publishing environments, provided by the major players in the field (examples are *Yahoo! Search Boss* and *Google’s Fusion tables*), so as to maximally ease the task of writing adapters.

The “vision” of Search Computing builds upon two new communities of users:

- **Content providers**, who want to organize their content (now in the format of data collections, databases, web pages) in order to make it available for search access by third parties. They will be assisted by the availability of a deployment environment facilitating at most their task, and will be provided with the possibility to register their data within a community. In this way, the “long tail” of content providers will see a concrete possibility of exploitation.
- **Application developers and/or expert users**, who want to offer new services built by composing domain-specific content in order to go “beyond” general-purpose search engines such as *Google* and the other main players. They will be assisted as well by the availability of visual tools facilitating at most their task, and will in addition find a deployment environment, either obtained by installing run-time components upon their servers, or - most interestingly - by finding servers already deployed within cloud computing architectures, where they will run their applications.

In the simplest scenario, the same person or organization may play the role of content and application provider, and offer to generic users the access to a specific content. In the most interesting and challenging scenario, application developers would act as the brokers of new search applications, built by assembling arbitrary resources, accessible through uniform service interfaces; some of them could be generic, world-wide, and powerful (e.g., general purpose search engines or geo-localization services), other resources could be specialized, local, and sophisticated (e.g., the “gourmet suggestions” about slow-food offers in given geographic regions). Moreover, expert users might visually compose queries, starting directly from resource networks, thus covering the gap in expressing a complex semantics to new generation search engine.

Most of the effort in Search Computing will then be dedicated to supporting content providers, application developers, expert users, and end users. We expect application developers to be aware of the resource networks and use visual tools for building applications with a high-level approach, consisting in using visual tools for selecting resource sub-networks and turn them into parametric query templates. The boundary between such actors is not completely sharp, as we do expect some users to be expert to the point of setting up an application themselves. In this vision, new business options open up for service providers and brokers, with appropriate licensing agreements regulating the rights to content access and the sharing of profits based

upon accountability of the click-through generated traffic or of actual committed transactions. This vision is compatible with the current models adopted by the major search engine companies (e.g., *Google* or *Yahoo*), which monitor the click-through traffic generated by advertising and sponsored links.

Some of the aspects considered in this research plan can be considered futuristic and difficult to accomplish, but Search Computing is a five-year project. This book reports the results of the first year of the project and illustrates our first moves to accomplish our vision. Four more years of investigation and development are ahead of us; the results of the project are available (now and throughout the project) on the project's website: [www.search-computing.eu](http://www.search-computing.eu).

**Acknowledgement.** Search Computing is a collective research effort, and its ambitious goals cannot be achieved without the essential contribution of many colleagues who are either working on it or are members of the Search Computing Advisory Board. At DEI (Politecnico di Milano): Adnan Abid, Davide Barbieri, Daniele Braga, Marco Brambilla, Alessandro Bozzon, Alessandro Campi, Sofia Ceppi, Sara Comai, Francesco Corcoglioniti, Emanuele Della Valle, Piero Fraternali, Nicola Gatti, Michael Grossniklaus, Mamoun Abu Helou, Andrea Maesani, Davide Martinenghi, Marco Masseroli, Maristella Matera, Davide Mazza, Emanuele Padula, Stefania Ronchi, and Marco Tagliasacchi; at DIG (Politecnico di Milano), Tommaso Buganza, and Roberto Verganti; internationally, Ricardo Baeza-Yates (Yahoo! Research, Barcelona), Fabio Casati and Florian Daniel (University of Trento), Georg Gottlob (Oxford University), Ihab Ilyas (University of Waterloo), Ioana Manolescu (INRIA, Paris), Norman Paton (University of Manchester), Gerhard Weikum (Max-Planck-Institut für Informatik), and Jennifer Widom (Stanford University).

The SeCo Project (Nov. 1<sup>st</sup> 2008 – October 31<sup>th</sup> 2013) is funded by an IDEAS Advanced Grant of the European Research Council.

# Chapter 2: Next Generation Web Search

Ricardo Baeza-Yates and Prabhakar Raghavan

Yahoo! Research  
Barcelona, Spain & Sunnyvale, USA  
rbaeza@acm.org, pragh@yahoo-inc.com

**Abstract.** In this chapter we provide our personal vision of what could be the next generation of Web search engines, outlining the main research challenges that derive from it. This vision is based on a single premise: people do not really want to search, they want to get tasks done. We motivate our work by the current trends in the Web and, in particular, Web search.

## 1 Introduction

Web search has become the starting point of many on-line user activities. The first generation of Web search engines faced two primary challenges: (1) to scale known information retrieval techniques to millions of documents, far beyond the capacity of search engines of the time; (2) to contend with document publishers that were diverse, non-uniform in quality/authority/style, and in some cases unreliable or even dishonest in their intent (known as Web spammers). The first of these challenges was addressed by employing coarse-grained parallel hardware to create robust, high-capacity computing services – these gave rise in time to what we now think of as cloud computing. The second challenge was addressed using various approaches from machine learning and link analysis, but the issue of spammers has never been completely solved. To this date, search engines fight an interactive battle with spammers using increasingly sophisticated techniques. As search engines devised better techniques to combat spam, they were able to adapt many of the same ideas to improve their ranking functions. Almost a decade ago, Google and other search engines began to perfect their responses to *navigational queries*: queries (such as “british airways”) whose goal is to take the user to a single target page (in this case, the home page of British Airways).

Navigational queries became the genesis of a new way of thinking about Web search: namely, the goal of the engine is not to retrieve relevant documents (the classical metaphor for three decades of information retrieval). Rather, the goal is to identify a user’s intent (navigating to a specific target page being one of many possible intents), and to synthesize a page that directly addresses the user’s intent. For instance, the query “Frankfurt temperature” is arguably not demanding a ranked list of websites any of which could provide the temperature in Frankfurt; rather, the user is best satisfied by a number that shows the current temperature in Frankfurt. Thus, Web search ceases to be about document

retrieval; rather, it is an interface for web-mediated user goals. The promise of these new engines is that instead of listing the top-ranked documents matching the user's query, they provide a new breed of search experiences. In the process, the user is saved the burden of culling documents from a results list and laboriously extracting the information buried within them.

The technical prowess of today's search engines – crawling, indexing, retrieval and ranking – will cease to be the differentiators for this next generation of search engines. Instead, the key to a better experience will come from the combination of the deeper analysis of content with the detailed inference of user intent. In Section 5 we explain how this would work in detail but the main ideas are: (1) in place of the indexing that search engines traditionally perform (mapping to each keyword those documents containing that keyword), we have a content analysis phase that spots *entities* (such as airlines, restaurants, professors and museums) in documents; (2) at query time we assign an *intent* to the user based on the query, the user's IP address and any other context that may be available (such as the GPS coordinates of the phone from which the query was launched); (3) we then retrieve entities matching the intent (say, Italian restaurants in the user's vicinity) and assemble a results page not of documents, but of matching entities and their attributes (for a restaurant these attributes would include reviews gathered from various websites, the menu and hours of operation from the restaurant's website, and location from a mapping service).

The organization of this chapter follows. In Section 2 we present the current state of the Web as a searchable data repository. In Sections 3 and 4 we present current Web and Search trends, respectively. In Section 5 we show how next generation search might work [29]. In Section 6 we outline several research challenges that are a direct consequence of current trends and our next generation search vision. We end the chapter with some final remarks in Section 7. During our exposition we use Yahoo!'s own research results as examples on the topics covered.

## 2 The Web

In the few years of its existence, the Web has become the largest repository of data created by the human kind. The number of static Web pages has been estimated to be in the tens of billions. Further, dynamic pages can be created in unbounded numbers (e.g. consider a Web calendar). Today, there are more than 230 million servers<sup>1</sup> and there are more than 680 million computers directly connected to Internet<sup>2</sup>. Hence, Web servers are nowadays a commodity, one for every three hosts.

Regarding the characteristics of the content, several studies indicate that today the Web is a reflection of society, and in particular of World economy. Other studies also show a high fraction of content redundancy (over 20%, see for example [9]). Hence, what we can find in the Web will range from popular repeated pages to many diverse unique pages.

---

<sup>1</sup> According to netcraft.com.

<sup>2</sup> According to the Internet Host Survey.

There are three main categories of Web data:

- Web content, mainly natural language text. The Web comprises hundreds of terabytes of text in several languages, sometimes with parallel translations. This content has been well described in [25].
- Link structure of the Web: links and anchor text encode semantic information and due to its large number is frequently used (see for example [17]).
- Usage data in the Web: human actions recorded in Web logs also encode semantic information and by volume is the largest resource available.

The structured data and semantics behind these (re)sources must be extracted by different techniques, which we explore later. Let us now describe the main characteristics of each case.

## 2.1 Content

Ramakrishnan and Tomkins [30] estimate the volume of content created every day. Their estimations suggest that the amount of content produced in the Web per day varies from 2Gb for professional Web content to 10Gb for User Generated Content (see next Section). They also estimate 3Tb per day for private content and 700Tb of generated text per day as an upper bound.

Regarding metadata, that is, data about data, they estimate per day rates ranging from 10Mb for reviews and comments to 100Mb for anchor text, with tagging in between at 40Mb. Implicit metadata coming from page views is estimated at 180Gb per day. Hence, metadata coming from usage is much larger than metadata coming from context, a fact that we will explore later.

## 2.2 Structure

The first (and last) study of the link structure of the whole Web was done by Broder *et al* [14] in 1999. They showed that the main part of the Web was the largest strongest connected component (SCC) of the link graph. The SCC had two attached components: pages with link paths to the SCC and pages connected to dead end link paths starting at the SCC. In addition there were many islands (groups of unconnected pages). From a practical point of view this implies that it is very easy to crawl the SCC and the pages linked from there, while the rest would need to be registered by the owners in the search engine. In the last decade, the rich link structure of the Web triggered many measures to evaluate the quality of Web pages based in links in the last, starting with PageRank and HITS.

## 2.3 Usage

Web usage involves any interaction of people with the Web. From a search engine point of view the most important interactions are querying and browsing. The frequency distributions of queries and URL clicks follow a power law, as many other variables that can be measured from the Web. Regarding queries, one important question is if the power law is (1) due to one group of people requesting



popular queries and other group of people asking unique queries (the long tail) or (2) due to all people asking both classes of queries. Goel *et al* have shown that the later case is what happens in practice [23].

Extracting and deducing information from usage data is one example of what today is called the wisdom of crowds [33].

## 3 Web Trends

### 3.1 User Generated Content

User generated content (UGC) is the content generated by users participating in experiences collectively referred to as Web 2.0. The Web 2.0 is associated with Web applications that facilitate interactive information sharing and collaboration, such as blogs, wikis, mashups and social networks. As we already mentioned, Ramakrishnan and Tomkins estimate the volume of UGC per day to be 10Gb. Although the average quality of UGC is not as good as editorial content, we believe that at the same level of content quality, the volume of UGC is larger than editorial content [2].

This trend has two main consequences on the Web:

- With increasing numbers of people creating and owning content each day, we have growing *fragmentation of ownership*. This in turn implies a democratization of Web content. While positive from a societal standpoint, this makes Web search more challenging due to the growing diversity of ownership of content.
- More content available means less time to consume content from each Web site. Hence, we have a *fragmentation of access*. This is a negative consequence, as the volume of “written-only” (and never read) content increases.

### 3.2 Social Networks

An important case of UGC that deserves special attention are social networks like Facebook and MySpace. The first surpassed 300 million members at the end of 2009 and has seen an impressive growth in the last year in many countries. The concept of different levels of access rights inside social networks, such as friends, friends of friends and geographical social sub-networks, contributes to the *fragmentation of the right to access*. That is, today we cannot simply talk about the public and the private Web, as the private Web is now fragmented and the accessible<sup>3</sup> Web depends on the user.

Another trend related to social networks is Twitter. Twitter is a micro-blogging and social network application that its founders call a real-time information network. To this real-time Web we must add data published in real-time coming from various sensor networks connected to the Internet.

The main question still unresolved is the viability of social networks as the classical advertising-based business model is not (yet) well established.

<sup>3</sup> In the sense of access not accessibility.

### 3.3 Web of Objects

The Web of Objects (WOO) is a new way of organizing Web content in terms of entities and relationships between them. The WOO is related to the Semantic Web initiative and the Open Linking Data project<sup>4</sup> is one of the best examples of what could be this Web in the future. Most linking techniques are based on content and link analysis, but we will see later that Web usage analysis can be an additional and powerful technique to improve the state of the art. It is important to notice that this is different from the Web of Things, which are physical objects that contain embedded devices connected to Internet that are integrated through the Web. A related topic is the recently defined Web of Concepts [21].

## 4 Search Trends

We next outline what we believe are the main current trends in search, in no specific order. For more information we refer the reader to [11,26].

### 4.1 Query Intent

As motivated in the introduction, search results have become more than a list of documents. Search engines are moving towards identifying the *intent behind the query* and enabling the user to complete a specific task. The first and most popular categorization of query intents in the Web was proposed by Broder [15]. He defined three classes of queries: *informational*, *navigational*, and *transactional*. Informational queries are those where the main goal is information as in traditional information retrieval. In navigational queries the goal is to find a Web site for browsing while in transactional queries, the goal is to execute interactive tasks such as downloading images or buying a product. Further, notice that the query intent can be ambiguous. For instance, consider someone seeking information on their favorite singer. Are they looking for the biography, the official Web site or a song? Broder estimated the percentage of the three classes of queries obtaining between 39%-48% for informational queries, 20%-25% for navigational queries, and 30%-36% for transactional queries. This taxonomy has been refined in several ways, but still is the most used one.

### 4.2 Open Search Ecosystem

Another trend is the open search ecosystem, where Amazon's OpenSearch and Yahoo!'s SearchMonkey are two major initiatives. OpenSearch is a suite of technologies that allow publishing of search results in a format suitable for syndication and aggregation. In this way, websites and search engines can publish search results in a standard and accessible format. On the other hand, SearchMonkey is a Yahoo! service which allows Web publishers to use structured data to make Yahoo! Search results more useful and visually appealing, and hence drive more

---

<sup>4</sup> URL: <http://esw.w3.org/topic/SweoIG/TaskForces/CommunityProjects/Linking-OpenData>

relevant traffic to their sites. Both, OpenSearch and SearchMonkey allow people to mash up user experiences based on metadata for user results. Microsearch [27] is an early example of this: you can see the metadata in the search and therefore are encouraged to add to it. There is a button next to every result called “Update metadata” which gives you instant feedback of what your metadata looks like.

## 5 How This Might Work

Our fundamental thesis is that the way to satisfy the user need underlying a query is to assemble entities – rather than documents – matching that need. To this end, we must identify specific types of entities that will be presented in response to user queries – for instance, we may decide that we will present entities of types restaurant, university, politician and automobile. For each entity type we may identify a schema that tells us what structured meta-data we associate with that entity type. For instance, the meta-data associated with the entity type restaurant could include its name, address, cuisine type, menu, reviews and opening hours.

Each of these entity types is presented in response to specific classes of queries relating to the entity type. Next we outline what is needed to accomplish this and the main research challenges, detailing them in Section 6.

### 5.1 Pre-processing

This approach demands a series of “pre-processing” steps that go beyond traditional parsing and inverted indexing of crawled content. There are two significant pre-processing steps:

1. Entity extraction: Here we extract, from all the documents in the index, all occurrences of entities of specified types. Thus in our running example, we may extract every instance of a restaurant, university, politician or automobile from each document in the index. The extraction itself can invoke any of several methods: identifying entities listed in a catalog (say, a list of all restaurants known to a publisher such as Zagat’s), regular expressions and other rules, or methods from machine learning. From this we can create a dictionary of all entities (of the selected types) present in any document, together with some meta-data about each occurrence.
2. Entity normalization: The above discussion raises the important issue of *normalization* – from all the occurrences of (say) *Pizzeria Roma* in many different documents, how many different physical restaurants can we infer? To put it differently, we must take all references to a single physical *Pizzeria Roma* and collate them, even if they occur in different documents.

The first step is performed as each crawled document is parsed. The second is done in a series of refinements in which we bucket the spotted entities into a dictionary of likely distinct entities, together with their meta-data. The net

of the two steps is a dictionary of all entities of the required types, together with our best estimate of the meta-data for each entity. Each of these steps is fraught with research challenges, covered in the next Section, in particular with what precision and recall can we cull entities of the various types from the noisy information on the Web?

## 5.2 Query Processing

The first goal of query processing is to assign an intent to the query at hand. Exactly how to specify the universe and language of expression of intents is a subtle and largely unsolved problem. To a first order, we may view an intent as a probability distribution over decompositions of the query into distinct entities.

For each combination of entity types recognized in the query (for instance, restaurant name plus metro area) we would need a templated response that prescribes the entity types to be presented in the results. Next, we retrieve and rank entities of the prescribed types; the ranking of entities is a major open problem.

# 6 Research Challenges

We now outline several research challenges associated to Web search. Some of them are related to trends already existing in the Web that will be needed by any Web search engine, while others are directly related to our next generation search vision.

## 6.1 Crawling

Due to the volume, diversity and rapid growth of Web data, recollecting the data in a timely manner has been always one of the main challenges behind Web search. In addition, as the link structure of the Web is not fully connected, that implies that not all Web data can be easily found. To this we have to add the hidden Web, for example, data behind forms in e-commerce sites. On top of that, the diversity of the Web poses the extra condition that a crawler must be tolerant to all possible types of errors.

Hence, crawling is a very hard dynamic scheduling problem. The resultant software must run in a parallel and distributed platform, which adds other restrictions as well as more challenging issues like synchronization and consistency.

## 6.2 Extracting and Ranking Entities

Entity extraction is one of the simplest natural language processing tasks, although in the context of the Web the problem is harder due to the volume of Web text and its quality (incompleteness, noise, truth or falsehoods, etc.) In the case of generic Web text is much more difficult to obtain the same quality, but this can be partially solved by relating content of other sources by using Web

mining as we mention later. We recognize that the vast majority of members of a given entity, say restaurant, are unlikely to be known to any publisher, so that we need rule-based and machine learning methods to discover entities not known to catalog publishers. Also, note that in general, a query need not be uniquely decomposed into entities; for instance, consider the query *chicago pizza new york*; is this a search for the restaurant “Chicago Pizza” in New York, or for the restaurant “Pizza New York” in Chicago? (There is also the third possibility that the query seeks Chicago-style pizza in New York, but this requires recognizing food types as entities.)

The problem of entity normalization becomes especially challenging in the face of noise (misspellings, abbreviations, typographical errors, etc), and the fact that most entity occurrences will likely be detected by machine learning and/or rules, rather than from a dictionary. For instance, along with a restaurant we may cull from a document its menu and two reviews; a different document may give us its opening hours and address. In other words, not all the available meta-data for an entity may come from a single Web page and in many cases, different Web pages might yield contradictory meta-data about an entity. Reconciling these is a part of entity normalization.

The next step is to rank information units of varying complexity and structure, in particular entities. Document ranking in today’s search engines is largely accomplished through machine learning, where document features are combined to produce scores (for each document on each query) that are close to those assigned by editors to selected example query-document pairs. In principle the same methodology could be used for ranking entities; however, the editorial assignment of scores to examples is much harder. An editor may reasonably be expected to score a document on its relevance to a query, judging some documents to be more relevant to a query than others. Ranking (say) pizzerias in San Francisco’s North Beach area is not a matter of retrospective editorial judgment, but rather a prognosis of likely user reactions to various pizzerias. Potentially, user reviews and other user generated content could – if suitably harnessed and spam-proofed – provide a scalable solution to this issue, at least for popular entities.

In this problem, our research has been focused in the quality/performance trade-off. We have shown that entity extraction can be done at state-of-the-art quality in linear time for good quality text such as Wikipedia [3]. We have also done work on the ranking of entities [34] or answers [32], based on semantic annotations. Some early demos of this research are Correlator, a new way to search the Wikipedia and find related entities, and Yahoo! Quest, a new way to explore Yahoo! Answers (see them at `{correlator,quest}.sandbox.yahoo.com`).

Another line of research has focused in information extraction in general [22], in particular for the case of evolving content [18]. The result is the PSOX information extraction system [13], that allows to do entity extraction taking in account the data source [31]. One additional semantic information to exploit in the future is time [1].

### 6.3 Query Intent

A major technique used to predict query intent is the analysis of Web query logs, or Web query mining, a topic that we explore in the next subsection. Recent research has focused on the automatic prediction of query intent [5,24]. Most studies are based on the application of machine learning techniques to different query attributes such as anchor-text word distribution in queries, click behavior, and query length, as well as related attributes such as the text of pages clicked on and the text of snippets associated with the results. Baeza-Yates *et al* [5] found that prediction accuracy was much higher for informational queries than for non-informational queries. In addition, as expected, they concluded that the intention of ambiguous queries is very hard to predict. Hence, further research on this problem is still needed.

One important attribute of intent is the physical location associated with it. For each query intent and its context (for example, location), we wish to change the result unit returned, improve the ranking of the results or show the results in a different way. Examples of this trend are structured or faceted results depending on the query intent.

One important issue when mining queries is privacy [19]. However, by aggregating all users that have the same intent, privacy risks are reduced as we are dealing with large groups of anonymous users and not specific individuals. In addition, aggregation helps boost statistical significance, as the intent distribution follows a heavy-tailed distribution and hence many user intents are rare. By aggregating people pursuing the same goal, we can personalize the experience of doing a task for more people.

### 6.4 Exploiting Web Queries

One of the most powerful sources of information in understanding query intent is what today is called *query mining* [4], that is the analysis of search engine query logs and the associated actions. The user behavior behind queries can relate content in many ways. Basically, if we can relate queries to each other, and also relate queries to content, we indirectly relate content, and hence entities. We can distinguish two types of query usage analysis of this flavor.

The first is based on *temporal causality*, that is queries are related because they are issued by the same person in sequence (that is, a logical query session). The strength of the relation is supported by the task that the person is trying to solve with a specific goal in mind, and by how many people issued a pair of queries in sequence. Hence, two pages are related if were clicked by two different queries that are related.

The second type of analysis is based on *behavioral causality*. For example, suppose one user asks the query  $q$  and clicks on page  $P$  and another user asks query  $r$  and also clicks on  $P$ . Then we can infer that queries  $q$  and  $r$  are related through the content  $P$ . In this case the strength of this relation grows with the number of people that performed this sequence of actions. Now, we can reverse the idea to obtain a *dual graph*: two pages are related if there is at least one query where one or more users clicked on those two pages.

In both cases we can infer a graph where the nodes are Web pages and two nodes are linked if some subset of queries related to each node are related. This graph can then be used to suggest possible related pages or equivalently, suggest dynamic hyperlinks while the user is browsing.

In addition, if we intersect the query graphs from both types of analysis, the result contains high quality signals because it is hard to have Web spam or noise that affects a given edge in both graphs. Other relations among queries can be obtained from the content and structure of pages or from the queries themselves [6].

If we can relate content, we can find semantic relations [7]. These relations can be used to automatically generate (pseudo)-semantic resources. Coupling them with open content resources (mainly coming from the Web 2.0), we create a *virtuous feedback circuit* to improve the Web and the data available in the Web. In fact, explicit and implicit folksonomies can be used to do machine learning without the need of manual intervention (or at least drastically reduce it), to improve semantic tagging [28].

One example of using query analysis to predict query intent is Yahoo!’s Search-Pad, where research queries are predicted to trigger a notepad that helps the user to keep annotations regarding the research topic of interest.

## 6.5 Results Page Layout

Where there are multiple intents (decompositions into entities) in the query, we have the additional challenge of laying out on the search results page the ranked entities for each intent. How do we optimize this presentation? In the traditional Web search interface, documents (deemed to be) matching the query are listed by decreasing order of score. A slightly more complex situation occurs in image (and certain forms of product) search, where the results are laid out in a two-dimensional “matrix” view; here the challenge is take the scores for the retrieved images and map them into positions on the matrix. While the commonest heuristic is to place the images in row-major order on the matrix, it is unclear whether this placement optimizes the user’s perception. This is because users do not typically scan the matrix in row-major order. The most general version of this problem: given ranked lists of entities matching the various intents in a query, how best can we use the screen area on the results page, to maximize the user’s utility? This challenging layout optimization problem can be decomposed in many ways into sub-problems, each of which gives rise to interesting research challenges.

## 6.6 Social Networks

From the point of view of Web search UGC and particularly social networks pose several new challenges, such as more real time content and more content that cannot be crawled. In addition, depending on the user need, the best answer may come from a different facet of the Web (static, dynamic, real-time, deep, semantic, etc.). Even further, we have the paradox that the answer a user may like to see should come from the facet of the Web that the user may never browse.

In this context several research problems arise: (1) crawling and searching real-time Web data while combating spam, (2) exploiting the underlying social networks to rank people and their UGC, and (3) using private data to infer signals about who may have the answer to a user’s need (e.g. expert search), without violating privacy restrictions.

## 6.7 Scalability

Scalability compounds all of these challenging research problems. In many cases we can improve results by just adding more data (e.g., in a machine learning algorithm related to entity extraction). This in turn leads to the research challenge of devising algorithms that strike a balance between speed and result quality. One major scalable programming technique used nowadays is the map-reduce paradigm through the Hadoop platform and cloud computing [20].

Scalability is also important at the infrastructure level, where data centers are each day larger [12]. One future alternative would be to transition from the current replicated centralized systems to truly distributed systems [10].

## 7 Final Remarks

In summary, the next generation of Web search must better extract meaning and semantics from all aspects of the Web – from the user’s query, to various forms of content. To a first approximation, eliciting meaning is a matter of eliciting structured entities from queries and content. Such extraction must then be combined with “more traditional” challenges such as spam filtering, ranking and scale.

Using Web usage to relate content can be thought as implicit crowd computing. That is, by searching the Web and clicking on results, users are helping computers to find similar content. This idea can be extended to many other applications and as more users use the Web, the *wisdom of crowds* becomes more powerful [33]. However, this effect creates or worsens other problems. An important example is: how to evaluate the results of these techniques, when each day the size of the ground truth data is relatively smaller (e.g. Wikipedia or the Open Directory Project). One possible solution is to use new sampling techniques that can assess with high probability the quality of the results.

Possible future next frontiers for search are semantic search and implicit search. Semantic search implies searching at the semantic and not the syntactic level. In implicit search, the query is implicit in the actions of the user and we will have an explicit or implicit information provision mode [16]. That is, instead of pulling information, information is pushed to us depending on the context. We can call this problem *contextual content delivery* and is applicable to any Web application. The final goal is to deliver the best possible content that a user would like to have in a given moment.

**Acknowledgements.** We are grateful for the helpful conversations with Andrew Tomkins and many Yahoo! colleagues.



## References

1. Alonso, O., Gertz, M., Baeza-Yates, R.: On the Value of Temporal Information in Information Retrieval. *ACM SIGIR Forum* 41(2), 35–41 (2007)
2. Anderson, C.: *The Long Tail: Why the Future of Business Is Selling Less of More*. Hyperion (2006)
3. Atserias, J., Zaragoza, H., Ciaramita, M., Attardi, G.: Semantically Annotated Snapshot of the English Wikipedia. In: *Proceedings of the 6th International Conference on Language Resources and Evaluation, LREC* (2008)
4. Baeza-Yates, R.: Applications of Web Query Mining. In: Losada, D.E., Fernández-Luna, J.M. (eds.) *ECIR 2005*. LNCS, vol. 3408, pp. 7–22. Springer, Heidelberg (2005)
5. Baeza-Yates, R., Calderón-Benavides, L., González-Caro, C.: The intention behind Web queries. In: Crestani, F., Ferragina, P., Sanderson, M. (eds.) *SPIRE 2006*. LNCS, vol. 4209, pp. 98–109. Springer, Heidelberg (2006)
6. Baeza-Yates, R.: Graphs from Search Engine Queries. In: van Leeuwen, J., Italiano, G.F., van der Hoek, W., Meinel, C., Sack, H., Plášil, F. (eds.) *SOFSEM 2007*. LNCS, vol. 4362, pp. 1–8. Springer, Heidelberg (2007)
7. Baeza-Yates, R., Tiberi, A.: Extracting Semantic Relations from Query Logs. In: *ACM KDD 2007, San Jose, California, USA, August 2007*, pp. 76–85 (2007)
8. Baeza-Yates, R., Mika, P., Zaragoza, H.: Search, Web 2.0, and the Semantic Web. In: Benjamins, R. (ed.) *Trends and Controversies: Near-Term Prospects for Semantic Technologies, January-February 2008*. *IEEE Intelligent Systems*, vol. 23 (1), pp. 80–82 (2008)
9. Baeza-Yates, R., Pereira, A., Ziviani, N.: Genealogical trees on the Web: A search engine user perspective. In: *WWW 2008: Proceedings of the 17th international conference on World Wide Web, Beijing, China*, pp. 367–376 (2008)
10. Baeza-Yates, R., Gionis, A., Junqueira, F., Plachouras, V., Telloli, L.: On the feasibility of multi-site Web search engines. In: *ACM CIKM 2009, Hong Kong, China, November 2009*, pp. 425–434 (2009)
11. Baeza-Yates, R., Ribeiro-Neto, B.: *Modern Information Retrieval*, 2nd edn. Addison-Wesley, Reading (2010)
12. Barroso, L.A., Hölzle, U.: *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. *Synthesis Lectures on Computer Architecture*, vol. 6. Morgan Claypool, San Francisco (2009)
13. Bohannon, P., Merugu, S., Yu, C., Agarwal, V., DeRose, P., Iyer, A., Jain, A., Kakade, V., Muralidharan, M., Ramakrishnan, R., Shen, W.: Purple SOX extraction management system. *SIGMOD Record* 37(4), 21–27 (2008)
14. Broder, A., Kumar, R., Maghoul, F., Raghavan, P., Rajagopalan, S., Stata, R., Tomkins, A., Wiener, J.: Graph structure in the Web: Experiments and models. In: *Proceedings of the Ninth Conference on World Wide Web, Amsterdam, Netherlands*, pp. 309–320. ACM Press, New York (2000)
15. Broder, A.: A taxonomy of Web search. *SIGIR Forum* 36(2) (2002)
16. Broder, A.: The Future of Web Search: From Information Retrieval Information Supply. In: Etzion, O., Kuflik, T., Motro, A. (eds.) *NGITS 2006*. LNCS, vol. 4032, pp. 362–362. Springer, Heidelberg (2006)
17. Chakrabarti, S.: *Mining the Web: Discovering Knowledge from Hypertext Data*. Morgan Kaufmann, San Francisco (2002)
18. Chen, F., Doan, A., Yang, J., Ramakrishnan, R.: Efficient Information Extraction over Evolving Text Data. In: *ICDE*, pp. 943–952 (2008)

19. Cooper, A.: A survey of query log privacy-enhancing techniques from a policy perspective. *ACM Transactions on the Web (TWeb)* 2(4) (2008)
20. Cooper, B., Baldeschwieler, E., Fonseca, R., Kistler, J., Narayan, P., Neerdaels, C., Negrin, T., Ramakrishnan, R., Silberstein, A., Srivastava, U., Stata, R.: Building a Cloud for Yahoo! *IEEE Data Eng. Bull.* 32(1), 36–43 (2009)
21. Dalvi, N., Kumar, R., Pang, B., Ramakrishnan, R., Tomkins, A., Bohannon, P., Keerthi, S., Merugu, S.: A Web of concepts. In: *PODS*, pp. 1–12 (2009)
22. Doan, A., Naughton, J., Ramakrishnan, R., Baid, A., Chai, X., Chen, F., Chen, T., Chu, E., DeRose, P., Gao, B., Gokhale, C., Huang, J., Shen, W., Vuong, B.-Q.: Information extraction challenges in managing unstructured data. *SIGMOD Record* 37(4), 14–20 (2008)
23. Goel, S., Broder, A., Gabrilovich, E., Pang, B.: Anatomy of the Long Tail: Ordinary People with Extraordinary Tastes. In: *Third ACM Conference on Web Search and Data Mining (WSDM)*, New York (2010)
24. Jansen, B.J., Booth, D.L., Spink, A.: Determining the user intent of Web search engine queries. In: *Proc. of the 16th international conference on World Wide Web*, pp. 1149–1150. ACM Press, New York (2007)
25. Kilgarriff, A., Grefenstette, G.: Introduction to the special issue on the Web as corpus. *Computational Linguistics* 29(3), 333–347 (2003)
26. Manning, C.D., Raghavan, P., Schütze, H.: *Introduction to Information Retrieval*. Cambridge University Press, Cambridge (2008)
27. Mika, P.: Microsearch: An interface for semantic search. In: *Proceedings of the Workshop on Semantic Search at the 5th European Semantic Web Conference, Tenerife, Spain (June 2008)*
28. Mika, P., Ciaramita, M., Zaragoza, H., Atserias, J.: Learning to Tag and Tagging to Learn: A Case Study on Wikipedia. *IEEE Intelligent Systems* 23(5), 27–33 (2008)
29. Raghavan, P.: The Future of Search. In: *5th Gilbane Boston Conference: Where Content Management Meets Social Media*, Boston (2008)
30. Ramakrishnan, R., Tomkins, A.: Toward a PeopleWeb. *Computer* 40(8), 63–72 (2007)
31. Shen, W., De Rose, P., Vu, L., Doan, A., Ramakrishnan, R.: Source-aware Entity Matching: A Compositional Approach. In: *ICDE*, pp. 196–205 (2007)
32. Surdeanu, M., Ciaramita, M., Zaragoza, H.: Learning to Rank Answers on Large Online QA Collections. In: *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, ACL-HLT (2008)*
33. Surowiecki, J.: *The Wisdom of Crowds*, Random House (2004)
34. Zaragoza, H., Rode, H., Mika, P., Atserias, J., Ciaramita, M., Attardi, G.: Ranking Very Many Typed Entities on Wikipedia. In: *CIKM 2007: Proceedings of the sixteenth ACM international conference on Information and Knowledge Management, Lisbon, Portugal (2007)*

# Chapter 3:

## Search for Knowledge

Gerhard Weikum

Max-Planck Institute for Informatics  
Saarbruecken, Germany  
weikum@mpi-inf.mpg.de

**Abstract.** There are major trends to advance the functionality of search engines to a more expressive semantic level. This is enabled by the advent of knowledge-sharing communities such as Wikipedia and the progress in automatically extracting entities and relationships from semistructured as well as natural-language Web sources. In addition, Semantic-Web-style ontologies, structured Deep-Web sources, and Social-Web networks and tagging communities can contribute towards a grand vision of turning the Web into a comprehensive knowledge base that can be efficiently searched with high precision. This vision and position paper discusses opportunities and challenges along this research avenue. The technical issues to be looked into include knowledge harvesting to construct large knowledge bases, searching for knowledge in terms of entities and relationships, and ranking the results of such queries.

### 1 Trends and Opportunities

It is widely believed that queries posed to Web search engines are very simple: one or two keywords to express the user's information need, and millions of matching results including many excellent hits, so that well-known ranking techniques can easily achieve high precision for the top-10 Web pages seen by the user. While this may indeed be true for the large mass of popular queries, each asked by many thousands of users, the picture is different for the long tail of individual queries about professional needs, rare hobbies, local music concerts, or personal health issues. Not only do these queries contain more keywords and return fewer results, but the user would often expect a concise answer with relevant facts rather than merely being pointed to potentially interesting Web pages. The following are examples of such advanced queries (we will later use some of these to illustrate technical challenges):

1. A student of natural history may want to know about explorers on river expeditions for some project work. A botanics student may be interested in succulents that grow in both America and Africa. While perhaps resembling quiz questions, this type of queries arises in the daily work of millions of university students.
2. As many people like watching TV game shows, true quiz questions may indeed be a use case as well. Which king was married to Eleanor of Aquitaine? Who was the last wife of Idi Amin? Who was the wife of the French president when Nicolas Sarkozy was born? If a search engine could automatically answer all these questions, a machine could win a million Euros in the popular quiz show "Who Wants to Be a Millionaire?"

3. This type of knowledge “factoid” questions can be extended into “list” questions that aim to retrieve comprehensive lists of persons with particular properties. Which Oscar winners are from Europe? Which scientists emigrated from Germany to America? Which French politicians are married to singers?
4. Sports is a topic with high query traffic by both professionals, such as journalists, and laymen like fans of particular teams or athletes. What is the highest number of points that any center player of the Los Angeles Lakers ever scored in the NBA league? Against which German soccer clubs did Real Madrid play in one of the European leagues or cups?
5. Finally, health is a topic of great importance in society. Information about diseases and pharmaceutical drugs becomes increasingly complex, while more and more (often elderly) people depend on accurate information. Examples of typical search requests are the following: Which drugs against flu symptoms or flu viruses can be safely used by children? Which of them can be taken while being pregnant? Could the H1N1 (swine flu) vaccine Pandemrix interfere with blood-pressure medications such as Metolazone?

Answering such queries requires a more semantic understanding of Web contents, lifting the interpretation of pages from the popular bag-of-keywords model to a level of *named entities* and *relationships between entities*. As results we expect ranked lists of entities or entity pairs that satisfy the relational conditions expressed in the query or question. For example, the sports query about Real Madrid should return clubs like FC Bayern Munich, Bayer Leverkusen, 1. FC Kaiserslautern, and so on.

We refer to this new level of Web querying as *search for knowledge*: facts on entities and relations, and not just Web pages. These kinds of advanced questions arise in the long tail of individual users’ needs for knowledge. In particular, they are central to the mission of many types of knowledge workers: scientists, students, journalists, market and media analysts, and so on. This demand is reflected in recent trends towards more powerful *semantic search engines* and *knowledge services* on the Internet.

Representatives of semantic search engines are *wolframalpha.com* which computes knowledge answers from a set of hand-crafted databases, *www.google.com/squared* which arranges search results in a tabular form with entities and attributes, *entity-cube.research.microsoft.com* which provides dynamically gathered facts about named entities, or *kosmix.com* which uses a large ontology for categorizing questions and identifying entities that are related to the user’s input. Examples of new kinds of knowledge services include *freebase.com* and *trueknowledge.com* which are compiling huge amounts of entity-relationship-oriented facts, the community endeavor *dbpedia.org* which is harvesting RDF subject-property-object triples from Wikipedia and similar sources, the *www.cs.washington.edu/research/textrunner/* project which aims to extract arbitrary relations from natural-language texts, the *sig.ma* engine which taps on “triplified” RDF data on the Web, as well as our own project *www.mpi-inf.mpg.de/yago-naga/* which integrates relational knowledge from Wikipedia with the WordNet taxonomy.

These services are enabled by the proliferation of knowledge-sharing communities like Wikipedia and by the advances in information extraction methods that can detect named entities and relational facts in both semistructured Web pages and natural-language text [17,28]. Harvesting Wikipedia and other high-quality sources has led to

very large knowledge bases with millions of named entities, systematically categorized into ten thousands of semantic classes, and more than 100 million facts about entities (with facts being instances of binary relations between entities). Harnessing these very large knowledge bases for answering knowledge questions and for boosting the semantic quality of Web search is a major opportunity that we see arising now.

Today's knowledge services and engines (mentioned above) are great steps in the right direction, but still have fundamental shortcomings. For example, wolframalpha can correctly answer the question "Who was French president when Nicolas Sarkozy was born?", but gives no answer to our full example query about the wife of that former French president. Likewise, it can compute Europe's longest river, but would not know the answer to the question "Which cities are located on Europe's longest river?". Similar limitations exist for all other systems as well. We can observe three major aspects on which current knowledge search falls short, leading to three technical challenges:

1. Knowledge bases are still too small and lack knowledge. This leads to the challenge of advancing the process of *knowledge harvesting*, discussed in Section 2.
2. The knowledge is available in principle, but the search methods are insufficient, lacking capabilities for multi-join queries and advanced inferencing. This leads to the challenge of extending *query processing* to deal with knowledge bases that are automatically built from Web sources. This issue is addressed in Section 3.
3. Large knowledge bases may contain noisy and incorrect information, producing many search results of highly varying quality. This leads to the challenge of appropriate *ranking models* for query results, in terms of entities and relationships rather than Web pages. This issue is the subject of Section 4.

In addition to these challenges, another critical aspect is to understand the question structure and interpret it as a formal query. The difficulty of this issue depends on the complexity of the question phrasing, and to what extent natural-language processing (NLP) can analyze and "understand" the question and map it into predicate-argument structures. State-of-the-art NLP, for example, dependency parsing, can cope fairly well with a wide variety of questions as long as the user avoids unnecessarily convoluted formulations. We thus disregard this aspect in this paper (although it remains an issue for NLP research for very sophisticated questions).

## 2 Challenge: Knowledge Harvesting

Knowledge search needs knowledge to start with. This could be given in the form of an explicit knowledge base, distilled from Web pages and compiled into an integrated collection of facts, or it could be in the form of semantically rich annotations of entities and relationships in the Web pages themselves. Both approaches require detection and extraction of relational facts in pages. The first approach - an explicit knowledge base - could bootstrap and ease the second method, by providing entities and seed facts to a broader extraction process. Therefore, we concentrate ourselves in the following on the construction of an explicit knowledge base, based on Web sources. We refer to this task as knowledge harvesting.

Comprehensive knowledge bases have been an elusive AI goal for many years. Ontologies and thesauri such as OpenCyc, SUMO, WordNet, or UMLS (for the biomedical

domain) are achievements along this route. But they are typically focused on intensional knowledge about semantic classes. For example, they would know that mathematicians are scientists, that scientists are humans (and mammals and vertebrates, etc.); and they may also know that humans are either male or female, cannot fly (without tools) but can compose and play music, and so on. However, the currently available ontologies typically disregard the extensional knowledge about individual entities: instances of the semantic classes that are captured and interconnected in the ontology. For example, none of the above mentioned ontologies knows more than a handful of concrete mathematicians (or famous biologists etc.). A comprehensive knowledge base should know all individual entities of this world (e.g., Nicolas Sarkozy), their semantic classes (e.g., Sarkozy isa Politician), relationships between entities (e.g., Sarkozy presidentOf France), as well as validity times and confidence values for the correctness of such facts.

Today, the best source for extensional knowledge is probably Wikipedia, providing a wealth of knowledge about individual entities and their relationships. This knowledge is latently embedded in the natural-language text of Wikipedia articles, but also exposed, to a limited extent, in semistructured elements like infoboxes, lists, and the category system for Wikipedia articles. The great success of such knowledge-sharing communities and the advances in automated information extraction (IE) methodology have enabled new ways of knowledge harvesting at large scale. IE comprises methods from pattern matching (e.g., regular expressions), linguistic analyses (e.g., part-of-speech tagging or dependency parsing), and statistical learning.

The DBpedia and YAGO projects [5,33] have pioneered massive fact extraction from infoboxes and categories in Wikipedia, to build large knowledge bases. DBpedia has emphasized recall by gathering all infobox attribute name-value pairs, at the risk of incorporating noise, inconsistent facts, and false results. YAGO, on the other hand, pursued the philosophy of high - near-human-quality - precision by employing database-style consistency checking on fact candidates. YAGO primarily gathers its knowledge by rule-based IE on the infoboxes and category system of Wikipedia, and reconciles the resulting facts with the taxonomical class system of WordNet [34]. Consistency checks include type constraints (e.g., *isMarriedTo* has type signature  $Human \times Human$ , *graduatedFrom* has type  $Human \times University$ ) and functional dependencies (e.g., for relation *isCapitalOf*,  $City \rightarrow Country$  is a function). The resulting knowledge base contains more than 2 million entities and 20 million facts, with at least 95 percent accuracy. YAGO has been incorporated into DBpedia and other projects. The Linking Open Data (LOD) initiative [9] provides extensive cross-linkage across the various knowledge bases, at the level of entity references.

For comprehensive knowledge bases, it seems unavoidable to tackle natural-language texts, in addition to harvesting semistructured data and structured databases. In Wikipedia, the by far largest fraction of facts is solely stated in the articles' textual bodies, and new, valuable knowledge is usually first produced in text form - in news and scientific publications. We can leverage existing, albeit limited knowledge bases to bootstrap extended forms of knowledge harvesting. To this end, we have developed the SOFIE system [35] for further growing the YAGO base in a high-quality, consistency-preserving manner. SOFIE parses natural-language documents, extracts new relational facts from them, and integrates the facts with the previously existing knowledge. SOFIE



uses logical reasoning on the existing knowledge and on the new knowledge in order to disambiguate words to their most probable meaning, to reason on the meaning of text patterns, and to take prescriptive constraints into account. This allows SOFIE to check the plausibility of new hypotheses and to avoid inconsistencies. For example, suppose that the prior knowledge base has seed facts about the *isMarriedTo* relation, so that we can automatically find potentially indicative patterns such as “*X and her husband Y*”, “*X, Y, and their children*”, or “*X has been dating with Y*”, and also new fact hypotheses such as (*Veronica, Silvio*), (*Carla, Silvio*), (*Carla, Nicolas*), and (*Carla, Mick*). By considering that certain fact candidates are mutually exclusive and patterns occur with different frequencies for different candidates, the statistical assessment of both patterns and fact hypotheses becomes much stronger.

SOFIE is based on mapping the intertwined tasks of pattern-goodness assessment, entity disambiguation, and fact-hypotheses selection into a weighted Max-Sat problem for which it provides a practical solver [35]. Alternative approaches include machine-learning methods such as Conditional Random Fields and Markov Logic Networks and also more scalable light-weight techniques, all of which have been successfully applied in projects at MSR Beijing [24,42], UW Seattle [10,19,40], UW Madison [16], and elsewhere (see [17,39] and references given there).

While the above approaches testify to the impressive progress that the research community has made on knowledge harvesting and the great potential for building comprehensive knowledge bases, there is still a long way to go. Advancing the state of the art faces a number of challenges, outlined in the following subsections.

## 2.1 Temporal Knowledge

So far we have simplified our knowledge-harvesting setting by assuming that facts are time-invariant. This is appropriate for some relation types, for example, for finding birthdates of famous people, but inappropriate for evolving facts, e.g., presidents of countries or CEOs of companies. In fact, time-dependent relations seem to be far more common than time-invariant ones. For example, finding all spouses of famous people, current and former ones, involves understanding temporal relations. Extracting the validity time of facts involves detecting explicit temporal expressions such as dates as well as implicit expressions in the form of adverbial phrases such as “last Monday”, “next week”, or “years ago”. Moreover, one often has to deal with incomplete time information (e.g., the begin of someone holding a political office but no end-of-term date given, although the person may meanwhile be dead), and with different time resolutions (e.g., only the year and month for the begin of the term, but the exact date for related events). In addition to the complexity of extracting temporal knowledge, this also entails difficult issues of appropriately reasoning about interrelated time points or intervals. For example, the constraint that each person has at most one legal spouse now becomes a more complex condition that the validity intervals of the *isMarriedTo* instances for the same person must be non-overlapping. Initial work on these issues includes [25,38,41].

## 2.2 Multilingual Knowledge

The English language represents a constantly decreasing fraction of the Web. China and the EU each have greatly surpassed the U.S. in the number of Internet users, and

other regions are expected to follow. Multilingual knowledge bases would address this development by providing entity labels in multiple languages and making the semantic connections between words and names in different languages explicit [1,15]. For this goal, the most natural approach seems to exploit multilingual labels of the interwiki links present in Wikipedia. For example, “Roma”, “Rome”, and “Rom” denote the Italian capital in three different languages (Italian, English, and German), but “Roma” is used in German with a very different meaning, referring to an Eastern European ethnic group. However, these links are noisy and cannot be blindly trusted. For example, the German article on “Momente (Stochastik)” (moments of a distribution) transitively leads to the Spanish article on “momento estandar” (standardized moments) which is related but not equivalent. Thus, establishing multilingual synonyms for entities is all but straightforward. Moreover, a full-fledged knowledge base should also be multicultural in the sense that it captures concepts that are unique or especially salient in specific languages while being absent or unremarkable in other cultures.

### 2.3 Multimodal Knowledge

With the proliferation of photo and video footage on the Web, a knowledge base would not be complete without multimodal data on individual entities (people, places, etc.) and important events (concerts, award ceremonies, soccer matches, etc.). While photos of celebrities are abundant on the Internet, they are much harder to retrieve for less popular entities such as notable computer scientists or regionally interesting churches. Querying the entity names in image search engines yields large candidate lists, but they often have low precision and unsatisfactory recall. Moreover, even for more prominent targets, it is desirable to have a diverse collection of photos (e.g., from different time periods), some of which might be rare and difficult to locate using search engines. In some cases, the ambiguity of the entity name dilutes the search engine results. An example is the Berkeley professor and former ACM president David Patterson. None of the top-20 Google image or Bing image results (as of August 2009) show him; most show the governor of New York (whose name is actually David Paterson). A first approach to overcome these problems is presented in [36], based on knowledge-driven query expansions and weighted ensemble voting on the results.

### 2.4 Active Knowledge

A knowledge base can never be complete and inevitably exhibits gaps. Suppose a user finds the biography of a singer interesting and then wants to find all songs and albums by this singer, including the latest ones. Crawling additional Web sites on music and extracting the missing data is often infeasible because of site restrictions and because the site’s information is continuously changing. Moreover, some knowledge is inherently ephemeral: for example, the current rating of a movie (by averaging user reviews) or the chart rank of a song. The approach to fill these gaps would be to harness the increasing number of Web services on music, movies, books, business directories, etc. This would require retrieving data from Web services on the fly, whenever the local knowledge base does not suffice to answer a user’s knowledge needs. Obviously, such a federated architecture entails several problems of high complexity: mapping search requests onto service interfaces, cost/benefit-oriented routing of queries to promising services, integrating results from different services, and more [12,27].



## 2.5 Diversity and Provenance

So far we have treated facts in a knowledge base as objective truth. This should indeed be the case for most facts, but some may be considered controversial. In these cases, we would like to cover diverse viewpoints, along with provenance information. For example, does smoking cause lung cancer? Despite strong evidence, there is no hard proof for a fact like *causes(smoking, lungCancer)*. Here the counter-view is only a minority. But majorities and minorities change over time. For example, back in the 17th century, the two competing facts *shape(earth, disc)* and *shape(earth, sphere)* were seen very differently than today. And the same holds for contemporary political affairs, suicide vs. murder theories, etc. Instead of letting mere statistics and machine learning decide on the currently most popular view, it would be better to keep diverse views and make diversity an asset rather than an impediment. But then, we need to capture also the provenance of conflicting statements (e.g., Catholic Church vs. Galileo Galilei for the earth-shape example). Of course, for undisputed, more or less universally accepted statements, noisy alternatives with low evidence should still be treated as incorrect. One of the challenges here is to tell which parts of a knowledge base require diversity and which ones are indeed objective. In general, the theme of knowledge diversity is virtually unexplored; the recently started EU project “Living Knowledge” is addressing some of the issues [23].

## 2.6 Scalability

Last but not least, knowledge harvesting also faces a formidable scalability challenge. Advanced methods for information extraction are computationally expensive, as they may require deep parsing of natural language and Markov-chain sampling for graphical learning models or dynamic programming on conditional random fields. Thus, we cannot easily run extractors on the entire Wikipedia text and expect high-quality results within a few hours. It is feasible to use more light-weight methods on surface text and simpler features for direct classifiers, but this is unlikely to yield high precision at reasonable recall. Recent projects along the lines of [2,19] scale up to high-throughput extraction with high recall, but they degrade in precision. To illustrate the performance requirements of a truly large-scale knowledge-harvesting system, consider all people who have a Wikipedia article, probably a few hundred thousands. We assume that these are prominent enough so that the Web somewhere, not necessarily in Wikipedia itself, has information about their spouses including dates of marriages, divorces, or becoming widowed. As a scalability benchmark, attempt to harvest the complete *isMarriedTo* relation for these people, complete with temporal information, so that the result has high precision, say at least 95 percent, and high recall, say at least 80 percent. Would any existing method be able to accomplish this benchmark task in one day?

## 3 Challenge: Query Processing

The DBpedia and YAGO knowledge bases represent all facts in the form of unary and binary relations: classes of individual entities, and pairs of entities connected by specific

relationship types. Other knowledge-harvesting projects such as freebase or trueknowledge have a similar flavor. This data model can be seen as a typed graph with entities and classes corresponding to nodes and relations corresponding to edges. It can also be interpreted as a collection of RDF triples with two adjacent nodes and their connecting edge denoting a (*subject, predicate, object*) triple, *SPO* triple for short.

### 3.1 Query Language

For querying such a knowledge base, it thus seems most natural to use the SPARQL language, the W3C standard for querying RDF data. For example, the two questions about Oscar winners from Europe and about French politicians who are married to singers can be expressed in SPARQL as follows:

```
Select ?x Where { ?x hasWonAward AcademyAward .
                  ?x isBornIn ?y . ?y locatedIn Europe . }
```

```
Select ?x Where { ?x isa Politician . ?x isCitizenOf France .
                  ?x isMarriedTo ?y . ?y isa Singer . }
```

These queries consist of conjunctions of elementary SPO search conditions, so-called triple patterns. Each triple pattern has one or two of the SPO components replaced by variables ?x or ?y, the dots between the triple patterns denote the conjunction, and using the same variable in different triple patterns denotes join operations. These queries may also be visualized as graph templates with node and edge labels that can be either constants (literals, class names, or relation names) to be matched by the result data or variables to be substituted by bindings to SPO values from the underlying data triples.

### 3.2 Schema-Free Querying

A salient feature of the RDF data model is that there is not necessarily a database schema for the SPO triples in a collection. SPARQL can easily process the entire spectrum from schematic to schema-less RDF datasets. For example, if the user did not know the name of the *isBornIn* property - the RDF counterpart of relation or attribute names in standard databases -, she could as well specify a wildcard property as follows:

```
Select ?x Where { ?x hasWonAward AcademyAward .
                  ?x ?r ?y . ?y locatedIn Europe . }
```

Now the variable ?r can bind to any property name in the dataset that contributes to satisfying the search conditions. For example, ?r could be satisfied by relations *isBornIn*, *birthPlace*, *isCitizenOf*, *hasLivedIn*, and so on. These include both semantically equivalent relations with different names, a typical situation when coping with heterogeneous data, and relations with different meanings. These forms of schema-relaxation or schema-ignorance are extremely useful when dealing with large sets of facts from best-effort knowledge harvesting.

An intriguing extension of SPARQL would be to allow transitive paths as a match [4]. This is similar to the XPath descendants axis, except that RDF data forms graphs

and not just trees, thus posing higher computational complexity. Even more generally, property names, or equivalently edge labels, in a query could be full-fledged regular expressions, which have to be matched by an entire path in the knowledge graph [22]. For example, suppose the *isBornIn* relation actually refers to cities and the *locatedIn* relation captures a city-county-state-country hierarchy. Further suppose that the user who asked about European Oscar winners does not care whether a candidate is born in Europe or is a citizen of a European country (and may be born elsewhere). We can use the triple pattern *?x ((isCitizenOf | isBornIn).(locatedIn)\*) Europe* to express this relaxed search condition. The semantics of such queries can be precisely defined, but the computational complexity of conjunctive queries with regular expressions for predicate names is challenging.

### 3.3 Temporal Querying

When dealing with time-variant knowledge, for example, the spouse, president, or CEO relations, temporal conditions on the validity of the facts of interest need to be expressible as well. For example, if we want to retrieve the wife of the person who was French president when Nicolas Sarkozy was born, we could think of phrasing this as:

```
Select ?x Where { ?x isMarriedTo ?y .
                  ?f:(?y isPresidentOf France) . NicolasSarkozy bornOn ?d .
                  ?f since ?t1 . ?f until ?t2 . ?d during [t1,t2] . }
```

Here we refer to fact identifiers like *?f* that can be used in other facts or triple patterns (a kind of reification). *?f* should be bound to the identifier of an SPO triple about a former president. The property names *since*, *until*, and *during* are temporal predicates between timepoints and time intervals, written as if they were stored properties although they should actually be functions dynamically evaluated on demand. This example shows the direction towards developing a query language for temporal knowledge. However, defining the precise semantics of such a language, characterizing its expressiveness and complexity, and developing reasonably efficient query processing techniques are open issues for further research.

### 3.4 Towards SPARQL Full-Text

Another extension of a SPARQL-style query language could be to combine triple patterns with keyword conditions. Whenever the user is not sure about whether interesting facts are captured by SPO triples in the knowledge base or merely expressed in textual form, a search engine should allow combinations of both search paradigms. For example, suppose we search for Italian Oscar winners who starred in western movies, perhaps with a story about the railroad and revenge. the user may not know how to express the condition about westerns in an SPO-based pattern. Is there a *genre* property in the knowledge base? Would it indeed use *western* as a corresponding object? To avoid these burdens, one would prefer simply giving keywords - but keywords within the context of a more precise triple pattern:

*Select ?x Where { ?x hasWonAward AcademyAward .  
                   ?x isCitizenOf Italy .  
                   ?x ?r ?m . ?m isa movie {western, railroad, revenge} . }*

Suppose that we further want to limit the results to actresses or composers, but preferably no male actors or directors. We could extend the third triple pattern by appropriate keywords, asking for matches to  $?x ?r ?m \{actress, composer\}$ . The entire query would then have a structural part and five keywords, but note that the five keywords are split into two groups and that each group refers to a particular triple pattern, not to the query as a whole. The keyword parts would have to be matched by facts that satisfy the corresponding triple patterns. With facts being automatically gathered from natural-language text documents and other Web sources, the system could use the text of the fact's origins as suitable context.

The outlined approach is close in spirit to languages like XQuery Full-Text [3] - except, and this is crucial, that we are dealing with RDF-style graph data and not with XML trees. Compared to keywords-only queries, the structural skeleton imposed by triple patterns is very helpful, too, as it gives a natural way of grouping keywords. For example, a query about French politicians who are related to Italian singers, could be expressed in a very relaxed manner as:

*Select ?x1, ?x2 Where { ?x1 ?r ?x2 .  
                           ?x1 ?p1 ?o1 {France, politician} .  
                           ?x2 ?p2 ?o2 {Italy, singer} . }*

This semantic grouping of keywords [20] is impossible with today's Internet search engines. (Note that having two separate phrase conditions "French politician" and "Italian singer" are not equivalent as these exact phrases may not occur in the desired matches.)

### 3.5 Programming and User Interfaces

Query languages of the above kind are suitable interfaces for programmers at the API level. They are not intended as end-user interfaces. But the API level is crucial to enable development of value-added applications on top of the knowledge services. With Web2.0 mashups and similar applications we already witness this transition into a *search-as-a-service* world. Here, advanced features of the query language and the richness of the underlying knowledge bases can blossom as enabling technology. In fact, it is conceivable and desirable that there will be many knowledge-search services with complementary knowledge bases, and that these would have to dynamically coupled in a search federation to accomplish application missions [12].

Notwithstanding this emphasis on the service API, better end-user interfaces are called for as well, as single services already provide a wealth of knowledge and can satisfy many user needs. Here, a formal language like SPARQL is clearly inappropriate, but the currently prevailing method of keyword queries is unsatisfactory as well. One avenue to explore is to live with keywords but aim to automatically impose structure on the keyword query by transforming it into a SPARQL-style representation. This would require detecting entities and relationships in the keyword query. Internet search engines already seem to apply such techniques to a small extent. For example, in the

query “Real Madrid soccer European league match German team”. Real Madrid could be identified as a named entity, and perhaps “German team” could be treated as a semantic class of entities. But the latter is already extremely difficult, and it would lack the connection to soccer anyway (and users would barely repeat a keyword by saying “German soccer team”). Moreover, these techniques would hardly be able to detect the relevant relationships between entities that the user is interested in. The cues for inferring that the query is about the *hasPlayedAgainst* relationship are way too implicit.

An alternative approach is to revive natural-language questioning [6] - the UI paradigm that question-answering (QA) services such as *answers.com* are using. As long as questions are phrased in a relatively straightforward way, modern NLP is able to parse them and understand the logical structure of the questions. Moreover, a full-fledged question contains better cues about the relations that should connect the entities or entity classes of interest, using verbal phrases and prepositions. For example, the question *Against which German clubs did Real Madrid play in one of the European leagues or cups?* makes it easier to map it to query conditions on the *hasPlayedAgainst* relation. Finally, natural-language questions are a naturally convenient way of expressing knowledge needs, especially compared to the alternative of stating a large number of keywords with carefully chosen order. Keyword queries are only the easier alternative if they are very short (the mass-user queries with one or two keywords). Last but not least, the success of smart phones such as iPhone, with built-in speech recognition, favors natural-language questions, as knowledge workers would want to speak in full sentences rather than uttering 5 to 10 keywords.

The envisioned change in search UI’s may resemble QA systems, but has a very different search architecture. Today’s QA systems would not map questions into formal queries, but rather aim to classify questions into fine-grained topics, or map them to the most similar natural-language question for which the system already has answers. So the query processing for knowledge search, as discussed in this paper, is different from current QA technology and would be a major step forward towards better knowledge answers.

## 4 Challenge: Ranking Model

Whenever queries return many results, we need ranking. For example, a query about *politicians who are also scientists* can easily yield hundreds of persons, solely based on information from Wikipedia categories. Even the more specific query about *French politicians who are married to singers* may overwhelm the users with possible answers. A meaningful ranking should consider the following two fundamental dimensions:

- *Informativeness*: Users prefer prominent entities and salient facts as answers. For example, the first query above should return politicians such as Benjamin Franklin (who made scientific discoveries), Paul Wolfowitz (a mathematician by training), or the German chancellor Angela Merkel (who has a doctoral degree in physical chemistry). The second query should prefer an answer like Nicolas Sarkozy over the mayor of a small provincial town. This ranking criterion calls for appropriate statistical models about entities and relationships.

- *Confidence*: We need to consider the strength or certainty in believing that the result facts are indeed correct. This is largely determined at the time when facts are harvested and placed in the knowledge base, and it can be based on aggregating different sub-criteria. First, the extraction methods can assign an *accuracy* weight to each fact based on the empirically assessed goodness of the extractor and the extraction target (e.g., rule-based for birthdates vs. linguistic for spouses) and the total number of *witnesses* for the given fact, i.e., the total frequency of observing the fact in the underlying Web sources. Second, the *provenance* of the facts should be assessed by considering the *authenticity and authority* of the sources from which facts are derived. PageRank-style link-graph-based models come to mind for authority ranking, but more advanced models of *trustworthiness* and entity-oriented rather than page-oriented importance are needed.

A good knowledge search engine should consider both criteria - informativeness and confidence - and combine them into a single scoring and ranking measure.

State-of-the-art ranking models in IR are based on *statistical language models*, *LM's* for short. They have been successfully applied to passage retrieval for question answering, cross-lingual search, ranking elements in semistructured XML data, and other advanced IR tasks [14]. An LM is a generative model, where a document  $d$  is viewed as a probability distribution over a set of words  $\{t_1, \dots, t_n\}$  (e.g., a multinomial distribution), and a query  $q = \{q_1, \dots, q_m\}$  with several keywords  $q_i$  is seen as a sample from this distribution. The parameters of the  $d$  distribution are determined by maximum-likelihood estimators in combination with advanced smoothing (e.g., Dirichlet smoothing). Now one can estimate the likelihood of query  $q$  for different candidate documents, and the one document that maximizes this likelihood should be the highest-ranked result.

#### 4.1 Entity Ranking

Recently, extended LMs have been developed for entity ranking in the context of expert finding in enterprises and Wikipedia-based retrieval and recommendation tasks [24,26,30,37]. For ranking entity-search results  $e$  to a keyword query  $q$ , one needs to compute  $P[q|e]$ . As an entity cannot be directly compared to query words, one considers the words in a Web page  $d$  that occur in a proximity window around the position from which  $e$  was extracted (i.e., the passage that contains  $e$ ). If  $e$  was independently extracted from  $K$  different pages, with empirically estimated accuracy  $\alpha_k$  from the  $k^{th}$  page  $d_k$ , the generalization of document-level LMs to entities needs to consider an  $\alpha_k$ -weighted aggregation of the  $K$  word distributions in the windows that contain the extracted entity  $e$ . Additional sophistication is needed for considering also an entity's attributes [24].

An alternative paradigm for entity ranking is to generalize PageRank-style link-analysis methods [13,21,32] to graphs that connect entities rather than Web pages. Statistical measures on the extraction process can be cast into edge weights for this purpose. For example, we could have entity-level links from soccer players such as Zidane or Beckham to their clubs such as Real Madrid, derived from hyperlinks between pages that contain the corresponding entities. This way an important player would transfer authority to his or her club, and vice versa. This line of models is useful, but appears to be more of an ad-hoc flavor compared to the principled LM approaches.

## 4.2 RDF Knowledge Ranking

The models discussed above are limited to entities – the nodes in an entity-relationship graph. In contrast, general knowledge search needs to consider also the role of relations – the edges in the graph – for answering more expressive classes of queries. Moreover, the discussed work on entity IR is still based on keyword search and does not consider structured query languages like SPARQL. Ranking for structured queries has been intensively investigated for XML [3], and, to a small extent, for restricted forms of SQL queries [11]. Ranking has also been studied in the context of keyword search on relational graphs (e.g., [7]). However, these approaches do not carry over to the graph-structured, largely schema-less RDF data collections and the expressive queries discussed in Section 3.

What we need for RDF knowledge ranking is a generalization of entity LM’s that considers relationships (RDF properties) as first-class citizens. We would like to estimate the likelihood that a possible answer generates the structured query. This can be broken down into likelihoods of SPO triples generating the corresponding triple patterns in the query. Recent work on the NAGA search engine [18,22] has addressed these issues and developed a full-fledged LM for ranking the results of extended SPARQL queries.

Here we merely illustrate the flavor and potential of this kind of ranking model by an example. Consider a question about soccer matches between an Italian and a German club (in one of the European leagues or cups). In SPARQL notation this may look like:

```
Select ?x Where {
  ?x hasPlayedAgainst ?y .
  ?x isa soccerClub . ?y isa soccerClub .
  ?x registeredIn Italy . ?y registeredIn Germany . }
```

Here the results are pairs of individual entities, Italian and German soccer clubs. A mere entity-ranking model would fail to yield the best results. It would favor important clubs like AC Milan or Juventus Torino on the Italian side and FC Bayern Munich or Hamburger SV on the German side. But one of the most spectacular matches ever was between Internazionale Milano and Borussia Mönchengladbach in the European Champions Cup in 1971. Borussia won 7:1, but the result was annulled because an Italian player was hit by a soft-drink can thrown by a spectator. So although both clubs are not the most prominent entities, the *hasPlayedAgainst* relationship between them is much more interesting than those between AC Milan and Bayern Munich, for example.

The example shows the potential but also the open challenges in entity-relationship ranking models. At this point, even the recent LM’s for RDF queries would not produce the ideal ranking for this particular example. In addition, efficiently evaluating the LM-based scores at query run-time in order to return the top-k best answers is an unsolved issue as well.

## 4.3 Personalization

The notion of informativeness is, strictly speaking, a subjective measure: an individual user wants to see an interesting, previously unknown but important, search result. This



calls for a *personalized ranking model* or at least a user-group-specific model. For example, we would often give children different answers than adults. For example, a query on “*gravitation black hole*” should probably not return the latest publications from physics journals or Einstein’s original manuscript to a 12-year-old student. As another example, consider a question about Oscar winners from Europe. In the non-personalized setting, we would rank people like Bruce Willis, Anthony Hopkins, Roman Polanski, or Ingrid Bergman as top results. But suppose the user has previously shown significant interest in music, expressed by her prior search-and-click behavior, e.g., browsing information on contemporary composers such as Philip Glass or Lisa Gerrard. Then the personalized search result should prefer people like Ennio Morricone, Hans Zimmer, or Javier Navarrete (all of which won Oscars for film music).

#### 4.4 Diversity

It is often desirable that the top-10 ranks of a search result are diversified. This issue has recently been investigated for Internet search, where the goal is to maximize the probability that the user will click on at least one result (or one of the sponsored ads) under the assumption that it is unlikely that she clicks more than one result. As a consequence, queries with ambiguous words such as “Real” should not be eagerly interpreted by using the most likely meaning only; instead, different meanings and diverse results should be reflected in the top results - for example, Real Madrid, Real Zaragoza (another Spanish soccer club), the Spanish royal family, real numbers, and so on.

For knowledge queries with entity-relationship-structured results, the notion of diversity is much less clear and largely unexplored. For example, when asking for important facts about a composer such as Ennio Morricone, the answer should not just focus on his popular music for western movies, but should also highlight his classical compositions, his work as a conductor, his awards, his family, his childhood, and so on. Query result diversity is a natural requirement for knowledge search, and poses many difficult issues to be explored.

## 5 Conclusion

This paper has presented opportunities for and challenges in moving from today’s keyword queries to a new quality level of semantic search for knowledge. We have discussed the statistical nature of knowledge harvesting and ranking entity-relationship-structured query results. Thus, the emphasis was on a *Statistical Web* setting. But there are also great assets in the *Semantic Web*, including hand-crafted, deep ontologies [31], and the *Social Web*, including tagging and cross-linking communities [8]. Connecting these different kinds of implicit and explicit knowledge sources opens up synergies and great opportunities towards the vision of large-scale knowledge management and search.

In a broad sense, knowledge harvesting may also be seen as an advanced form of information integration, and searching knowledge then is related to query processing over federations of data sources and services, as pursued by the “Search Computing” paradigm [29]. On closer look, however, these two paradigms are complementary: knowledge harvesting is focused on information around individual entities whereas search computing is more schema-oriented and focused on mappings and matchings



between the structures and types of entire sources. These two themes can greatly fertilize each other and may, in the long run, converge to a unified notion of semantic and structured search over arbitrary Web sources.

## References

1. Adar, E., Skinner, M., Weld, D.S.: Information Arbitrage across Multi-Lingual Wikipedia. In: WSDM 2009 (2009)
2. Jain, A., Ipeirotis, P.G., Doan, A., Gravano, L.: Join Optimization of Information Extraction Output: Quality Matters! In: ICDE 2009 (2009)
3. Amer-Yahia, S., Lalmas, M.: XML Search: Languages, INEX and Scoring. SIGMOD Record 35(4) (2006)
4. Anyanwu, K., Maduko, A., Sheth, A.P.: SPARQ2L: Towards Support for Subgraph Extraction Queries in RDF Databases. In: WWW 2007 (2007)
5. Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R., Ives, Z.G.: DBpedia: A Nucleus for a Web of Open Data. In: Aberer, K., Choi, K.-S., Noy, N., Allemang, D., Lee, K.-I., Nixon, L.J.B., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., Schreiber, G., Cudré-Mauroux, P. (eds.) ASWC 2007 and ISWC 2007. LNCS, vol. 4825, pp. 722–735. Springer, Heidelberg (2007)
6. Baeza-Yates, R.A., Ciaramita, M., Mika, P., Zaragoza, H.: Towards Semantic Search. In: Kapetanios, E., Sugumaran, V., Spiliopoulou, M. (eds.) NLDB 2008. LNCS, vol. 5039, pp. 4–11. Springer, Heidelberg (2008)
7. Bhalotia, G., Hulgeri, A., Nakhe, C., Chakrabarti, S., Sudarshan, S.: Keyword Searching and Browsing in Databases using BANKS. In: ICDE 2002 (2002)
8. Breslin, J.G., Passant, A., Decker, S.: The Social Semantic Web. Springer, Heidelberg (2009)
9. Bizer, C., Heath, T., Idehen, K., Berners-Lee, T.: Linked Data on the Web (LDOW 2008). In: WWW 2008 (2008)
10. Cafarella, M.J.: Extracting and Querying a Comprehensive Web Database. In: CIDR 2009 (2009)
11. Chaudhuri, S., Das, G., Hristidis, V., Weikum, G.: Probabilistic Information Retrieval Approach for Ranking of Database Query Results. ACM Trans. Database Syst. 31(3), 1134–1168 (2006)
12. Ceri, S.: Search Computing. In: ICDE 2009 (2009)
13. Chakrabarti, S.: Dynamic Personalized Pagerank in Entity-Relation Graphs. In: WWW 2007 (2007)
14. Croft, W.B., Metzler, D., Strohman, T.: Search Engines - Information Retrieval in Practice. Addison-Wesley, Reading (2009)
15. G.: Towards a Universal Wordnet by Learning from Combined Evidence. In: CIKM 2009 (2009)
16. De Rose, P., Shen, W., Chen, F., Lee, Y., Burdick, D., Doan, A., Ramakrishnan, R.: DBLife: A Community Information Management Platform for the Database Research Community. In: CIDR 2007 (2007)
17. Doan, A., Gravano, L., Ramakrishnan, R., Vaithyanathan, S. (eds.): Special Issue on Information Extraction. SIGMOD Record, vol. 37(4) (2008)
18. Elbassuoni, S., Ramanath, M., Schenkel, R., Sydow, M., Weikum, G.: Language-model-based Ranking for Queries in RDF-Graphs. In: CIKM 2009 (2009)
19. Etzioni, O., Banko, M., Soderland, S., Weld, D.S.: Open Information Extraction from the Web. CACM 51(12) (2008)

20. Graupmann, J., Schenkel, R., Weikum, G.: The SphereSearch Engine for Unified Ranked Retrieval of Heterogeneous XML and Web Documents. In: VLDB 2005 (2005)
21. Hristidis, V., Hwang, H., Papakonstantinou, Y.: Authority-based Keyword Search in Databases. *TODS* 33(1) (2008)
22. Kasneci, G., Suchanek, F.M., Ifrim, G., Ramanath, M., Weikum, G.: NAGA: Searching and Ranking Knowledge. In: ICDE 2008 (2008)
23. Living Knowledge, <http://livingknowledge-project.eu/>
24. Nie, Z., Ma, Y., Shi, S., Wen, J.-R., Ma, W.-Y.: Web Object Retrieval. In: WWW 2007 (2007)
25. Pasca, M.: Towards Temporal Web Search. In: SAC 2008 (2008)
26. Petkova, D., Croft, W.B.: Hierarchical Language Models for Expert Finding in Enterprise Corpora. In: ICTAI 2006, pp. 599–608 (2006)
27. Preda, N., Suchanek, F.M., Kasneci, G., Neumann, T., Ramanath, M., Weikum, G.: ANGIE: Active Knowledge for Interactive Exploration. *PVLDB* 2(2) (2009)
28. Sarawagi, S.: Information Extraction. *Foundations and Trends in Databases* 2(1) (2008)
29. SeCo: Search Computing, <http://www.search-computing.it/>
30. Serdyukov, P., Hiemstra, D.: Modeling Documents as Mixtures of Persons for Expert Finding. In: Macdonald, C., Ounis, I., Plachouras, V., Ruthven, I., White, R.W. (eds.) ECIR 2008. LNCS, vol. 4956, pp. 309–320. Springer, Heidelberg (2008)
31. Staab, S., Studer, R.: Handbook on Ontologies, 2nd edn. Springer, Heidelberg (2009)
32. Stoyanovich, J., Bedathur, S.J., Berberich, K., Weikum, G.: EntityAuthority: Semantically Enriched Graph-Based Authority Propagation. In: WebDB 2007 (2007)
33. Suchanek, F.M., Kasneci, G., Weikum, G.: YAGO: a Core of Semantic Knowledge. In: WWW 2007 (2007)
34. Suchanek, F., Kasneci, G., Weikum, G.: YAGO: A Large Ontology from Wikipedia and WordNet. *Journal of Web Semantics* 6(39) (2008)
35. Suchanek, F., Sozio, M., Weikum, G.: SOFIE: a Self-Organizing Framework for Information Extraction. In: WWW 2009 (2009)
36. Taneva, B., Kacimi, M., Weikum, G.: Gathering and Ranking Photos of Named Entities with High Precision, High Recall, and Diversity. In: WSDM 2010 (2010)
37. Vallet, D., Zaragoza, H.: Inferring the Most Important Types of a Query: a Semantic Approach. In: SIGIR 2008 (2008)
38. Wang, Y., Zhu, M., Qu, L., Spaniol, M., Weikum, G.: Timely YAGO: Harvesting, Querying, and Visualizing Temporal Knowledge from Wikipedia, Demo Paper. In: EDBT 2010 (2010)
39. Weikum, G., Kasneci, G., Ramanath, M., Suchanek, F.: Database and Information-Retrieval Methods for Knowledge Discovery. *CACM* 52(4) (2009)
40. Wu, F., Weld, D.S.: Automatically Refining the Wikipedia Infobox Ontology. In: WWW 2008 (2008)
41. Zhang, Q., Suchanek, F.M., Yue, L., Weikum, G.: TOB: Timely Ontologies for Business Relations. In: WebDB 2008 (2008)
42. Zhu, J., Nie, Z., Liu, X., Zhang, B., Wen, J.-R.: StatSnowball: a Statistical Approach to Extracting Entity Relationships. In: WWW 2009(2009)

## **Part II**

# **Technology Watch for Search Computing**

## **Introduction to Part II**

### **Technology Watch for Search Computing**

The second part of the book presents surveys of the technologies providing foundations to search computing. These chapters offer state-of-the-art and research trends within strongly related fields of research, useful both for setting the theoretical premises for search computing, and for providing a technological framework for building search computing systems and applications.

Chapter 4 includes the analysis and classification of **search systems**, which are facing a time of extremely rapid development. The study discusses a methodological framework for clustering search systems within categories; as a byproduct, the study detects decision variables and search engine features which are most likely to produce innovation and value in the search engine industry.

Chapter 5 deals with **mashup languages and systems**, a new way of describing computer processes through visual abstractions; mashup interfaces are very relevant to search computing, given that queries aim at the efficient interconnection of search engines and are primarily addressing expert users or developers.

Chapter 6 deals with **data extraction on the Web**, describing mechanisms for extracting information which is available on Web pages and putting it into repositories, by capitalizing on the experience of the Lixto project; data extraction technology is essential for building and exposing data services. This chapter deals with monitoring Web content and alerting users when information is updated.

Chapter 7 focuses on **data spaces** as a new concept for gluing loose and flexible approaches to data management, which give rise to a variety of new services for exposing data to wider usage; indeed search computing primarily pursues a data-driven approach to search service compositions and takes advantage of flexible technologies for exposing data sources.

Chapter 8 presents a review of **search technologies for multimedia content**, by showing the processes and tools for augmenting audio and video content with meta-data, so as to facilitate search upon multimedia content and its integration within search results.

# Chapter 4:

## The Search Engine Industry

Tommaso Buganza<sup>1</sup> and Emanuele Della Valle<sup>2</sup>

<sup>1</sup> Dipartimento di Ingegneria Gestionale, Politecnico di Milano, 20133 Milano, Italy

<sup>2</sup> Dipartimento di Elettronica e Informazione, Politecnico di Milano, 20133 Milano, Italy  
{tommaso.buganza, emanuele.dellavalle}@polimi.it

**Abstract.** In this chapter we present the main trends in the search engine industry. Being such industry technology based, its dynamics can be assessed by applying theories such as (a) dominant design, (b) complementary assets, (c) product and service architecture and (d) disruptive technologies. We dedicate the first section of this chapter to reviewing such literature and explaining how to apply it to identify trends in the search engine industry competition. As preliminary result we position the search engine industry among those that are probably entering in a new fluid phase. In this industry the Google architecture already emerged as dominant design, but after 2005 many new players entered the market (e.g. Cuil, Kosmix, Powerset, Wolfram Alpha, Bing) and most of them are not following the dominant design but are really trying to propose something radically new. Then, we present the data gathering tool we build to use analyze a sample of 26 search engines. In particular, we describe the dimensions, relevant to study the search engine industry, and the metrics for measuring the features of different search engines along those dimensions. We consider three types of metrics: (a) user based – what the user can perceive and act upon; (b) machinery related – what the search engine does internally; and (c) business model oriented – what makes the business profitable. Then we analyze the data using three methods: principal component analysis, two steps cluster analysis, and post hoc analysis on the business models categorization. We close the chapter discussing the results of our analysis.

### 1 Problem Setting

It is commonly recognized that the search engine industry started in 1990 with the release of the very first tool used for searching on the (pre-web) Internet: Archie. Since then, many different companies launched their own solutions in order to fulfill the market need for searching on the web. Search engines have become a usual service for the large majority of us to the point that more than 13 billion searches are made every month just in the US<sup>1</sup>. As for many Internet related industries, though, the companies operating on this market found it difficult to transform the value they had into real cash flows. Still, nowadays the search engine industry is probably one of the most significant markets in the Internet world. The main revenue streams are related to marketing activities and, just in the US, the market dimension is around \$11 billion

---

<sup>1</sup> [http://www.comscore.com/Press\\_Events/Press\\_Releases/2009/3/US\\_Search\\_Engine\\_Ranking](http://www.comscore.com/Press_Events/Press_Releases/2009/3/US_Search_Engine_Ranking)

	2006	2007	2008	2009	2010	2011
<b>Search</b>	6,799	8,624	11,000	12,935	14,906	16,590
<b>Display ads</b>	3,685	4,687	5,913	6,663	7,500	8,190
<b>Classified</b>	3,059	3,638	4,675	5,493	6,281	6,930
<b>Rich media</b>	1,192	1,755	2,613	3,575	4,463	5,481
<b>Other</b>	2,144	2,696	3,299	3,834	4,350	4,809
<b>Total</b>	16,879	21,400	27,500	32,500	37,500	42,000

**Fig. 1.** Online advertisement spending in millions by format (source [30])

in 2008 (the estimate is still growing to nearly \$13 billion in 2009 and over \$16 billion in 2011) (see Figure 1). In 2008 Google raked in 81% of US paid search advertising. Number two, Yahoo!, collected a mere 7% share, while everyone else split 12% of the pie. That's still a lot. With over \$16 billion going to search engine advertising in 2011, that 12% stake equals nearly \$1.9 billion: even a small slice represents significant revenue.

In just ten years from its birth the search engine industry went through all the phases that are typical for technology-based markets as described by Abernathy and Utterback [3,1] since 1978. Those markets characterized by strong technological discontinuities, like the birth of Internet, normally see an initial phase in which many different companies try to make their proposal to the market to better fulfill the existing or latent needs. During this phase, normally called “fluid phase” the market is characterized by high levels of product innovation and the market doesn't seem able to select the “best product”. In many cases some products have some good features but none is able to encompass all of them. The search engine industry lived this phase from its birth to the year 2000. In these years many different search engines with different technologies, services and market approaches struggled to conquer the market. Just to give some examples: Infoseek (1994) [33] offered web host pages, HotBot (1996) claimed to update its search database more often than its competitors, Webcrawler [34] (1994) was the first search engine to provide full text search, Alta-vista (1995) which introduced the multi-threaded crawler (Scooter) and an efficient search back-end running on advanced hardware, Yahoo! (1995) which was powered by Inktomi [36] and of course Google (1998) which introduced PageRank [31,32].

According to the Abernathy-Utterback model [1], this fluid phase is normally closed by the affirmation of a Dominant Design (DD), that is the solution winning on the market. It is important to notice that the dominant design is not always also the best technology architecture, as the famous case of VHS vs. Betamax [4] clearly shows. Still, the dominant design involves also the rise of the technological dominant architecture and the competitors are forced to cope with it as a standard.

Tushman et al. [2] described the main characteristics of a dominant design, among them we have:

- It is the architecture winning on the market having more than 50% of share. Looking at current data in terms of market share (see Figure 2), Google has approximately a share of 80%.

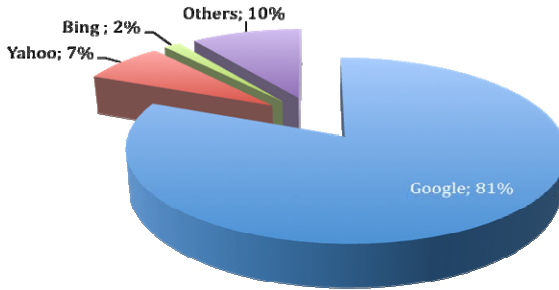


Fig. 2. Main Players' Market share in 2009 (marketshare.hitslink.com)

- It is the archetype of the product in both the user and the designer imagination. Before Google, the leading search engines (Yahoo!, Lycos [35], Altavista etc.) were converging towards a portal offering. The search functions were in bundle with other services in order to provide prepackaged information to the final users. Google offered a completely different approach focusing (at least at the beginning) just on the search functions. In other words they were able to reduce the list of requirements to be satisfied and to definitely define the set of functions to be offered. This approach turned to be standard for the industry and since then, the large majority of new search engines shared it (eg. Ask, Cuil, Powerset, ChaCha, and even Bing). Also from the technical design point of view Google set a new architecture. For example Yahoo! provided search services based on Inktomi's search engine until 2000, when it switched to Google's search engine (until 2004).
- It gives an answer to the need of a large number of people. Doubtlessly Google is able to give an answer to the large majority of queries at the web level. Still, the addition of more focused services like images, maps, scholar etc, made it fit with almost all the search needs.
- It normally freezes the socio-economic context. The leading position in the market was taken by many different companies before the 2000. Lycos was the most visited site in 1995 and Altavista the year after. Still, after the year 2000 the leading position of Google was not really challenged anymore. In Figure 3 we can observe that Google has a constant 80% of marketshare, followed by Yahoo! with a 7%. The crystallization of the market share, the concentration of the market (two companies alone have almost 90% of market share) and the introduction of several incremental innovations (maps, scholar, images, news, video etc) on top of the same technological/business model, are all symptoms of the presence of a strong Dominant Design.

Given the above considerations we can conclude that in the search engine industry the rise of Google allowed to define a Dominant Design in terms of technology, market approach ad business model and that the leadership of such a dominant design shows to be strong even today.

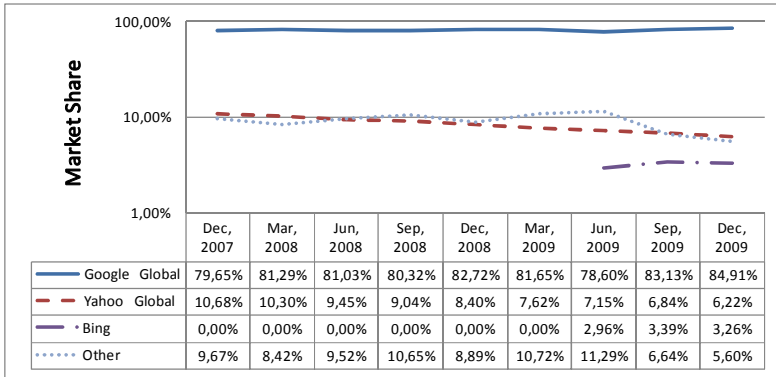


Fig. 3. Search Engine Market Share 2007-2009 (marketshare.hitslink.com)

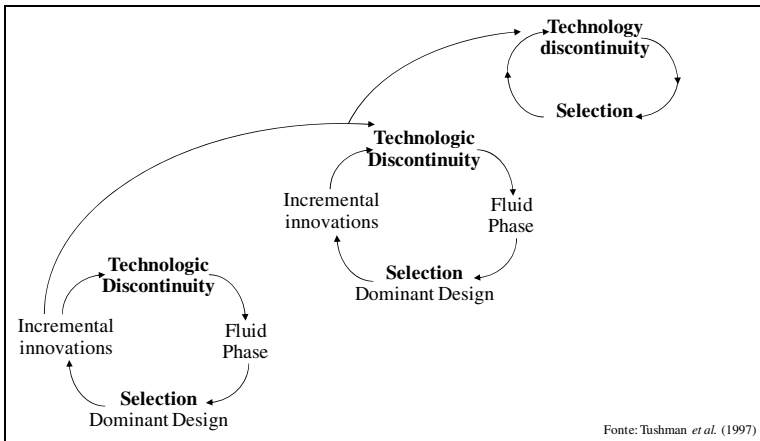


Fig. 4. Cyclical model for Innovation (source [2])

Still, the innovation management literature showed that no Dominat Design can last forever. Tushman et al. [2] showed that the market dynamics in technology-based environments are cyclical. When a solution wins on the market, it opens an era of incremental innovations, the solution leaders turn to be incumbent on the market and their leadership is hardly attackable unless something changes the situation dramatically. This dramatic change is often given by a new technological discontinuity. There are many examples of mature and static markets that were hardly shaken by technological discontinuities. Normally these discontinuities make the game start again; they open a fluid phase that ends once again with the emergence of a new dominant design (see Figure 4).

Examples of such dynamics are the switch from vacuum tubes to transistors, or from oil based illumination to light bulbs. In more recent times we observed the switch from analog to digital technologies that heavily shook the tv-set, the cameras and the music industries.



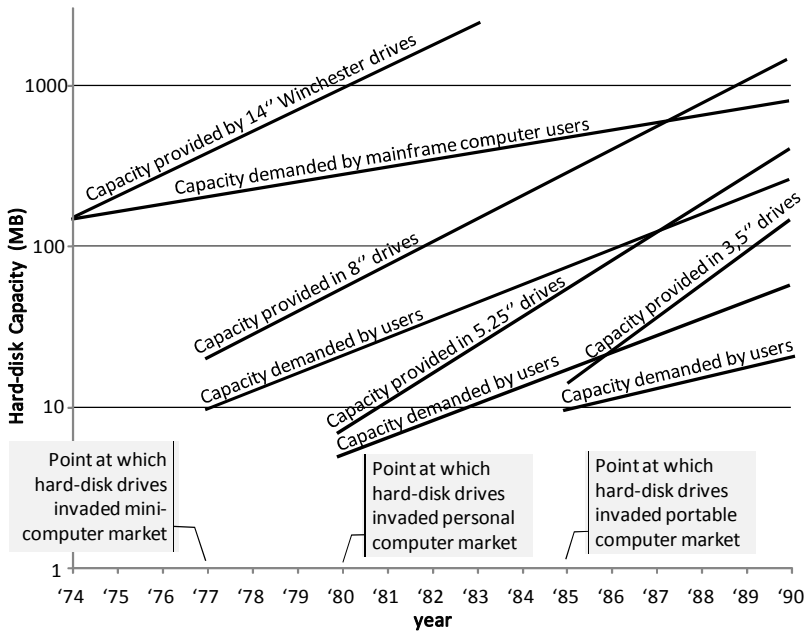


Fig. 5. Disruptive Technologies (source: Bower and Christensen [5])

In many cases the company that were market incumbents with the old technologies and that introduced the dominant design, are unable to face the technology switch on time and they lose their leadership position. It happened to RCA (vacuum tubes) vs. Motorola Fairchild and Texas Instruments (Transistors) as well as to Sony (Walkman) vs. Apple (iPod). Still, there are also cases in which the technological change was surfed by the incumbent companies that went through it without losing their leadership (or even strengthen it) as in the case of Windows in the switch from 3.x to 95/NT or on the case of Nikon and Canon in the switch from SLR cameras to Digital SLRs.

To define precisely what are the main reasons that distinguish the cases of falling leaders from those of surviving leaders is very complex and debated task. It is not clear, indeed, why some leaders are able to see the threat in advance and react to it while some others don't. Bower and Christensen [5], studying the disk drive industry from 1976 to 1992, observed that mainframe computers' needs were satisfied for a long time only by 14" drives, but starting from 1988 the 8" drivers (lighter and cheaper) matched the mainframe requirements and became the new dominant design. The same dynamic happened for 5.25" Drives vs. 8" drives and for 3.5" drives vs. 5.25" drives respectively for Minicomputers and Desktop PCs.

In all these cases the companies leading the markets with a technology were replaced by other companies bringing the new architecture. The strangest thing, though, is that the replacing technology was not new to the world, it was largely used in similar and parallel markets with lower performance requirements. For this reason they are defined Disruptive Technologies. The main explication why incumbents do not pay enough attention to disruptive innovations is because they are too close to their customers trying to fulfill their needs and to control their direct competitors. In this

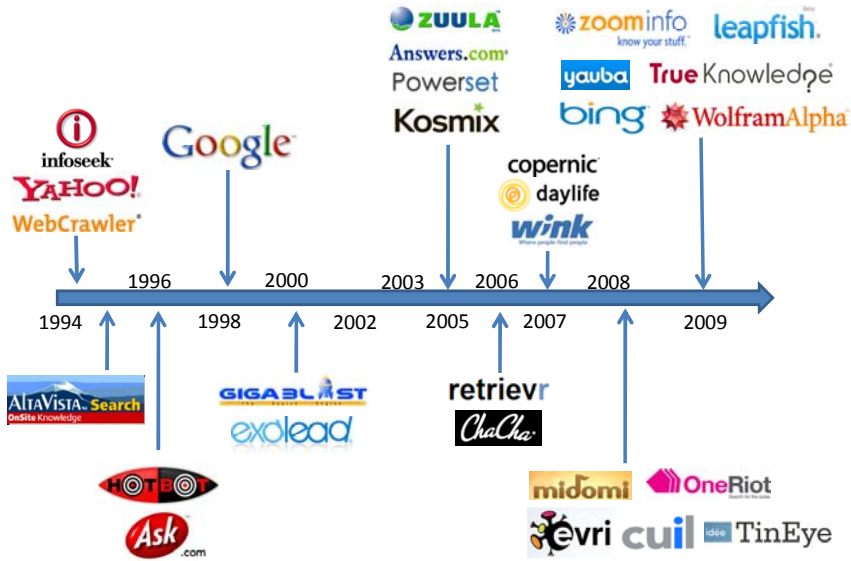


Fig. 6. Main new entrants on the market during the last years

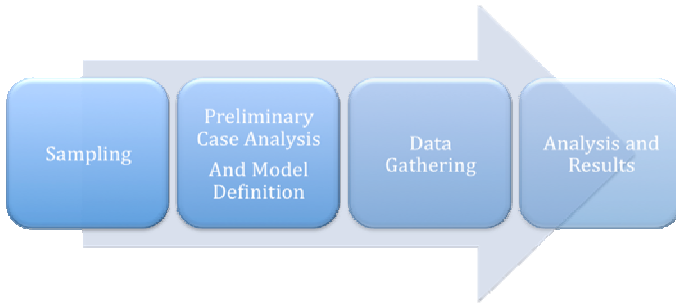
way, they are not focused to look outside their markets to intercept in advance those technologies that are not suitable for their market now but could become it sooner or later. Nevertheless, some companies competing in high dynamic markets seem to have learnt how to search for disruptive technologies in advance. It is the case of Microsoft that, in order not to be surprised by unexpected technologies, is ready to invest in potentially interesting technologies knowing that many of them will be not worth of it [6].

Finally we can observe that something is happening in the search industry sector. If it is true that the Google leadership it is not currently challenged, it is also true that when a dominant design is strongly in place the number of new entrants is normally low as well as the number of technological alternatives proposed to the market. On the contrary we observe that many new players entered the market since 2005 (e.g. Cuil, Kosmix, Powerset, Wolfram Alpha, Bing) (see Figure 6) and in the recent past much rumor was done by Bing and Wolfram Alpha. Many of these new players are not following the dominant design but are really trying to propose something radically new. This dynamic is typical of the fluid phase more than of a dominant design one. Moreover important strategic maneuvering are in place, like the rumors about Microsoft and Yahoo! merging.

All these considerations bring to draw a main question: *are we living the disruptive era of the Google architecture?*

## 2 A Method to Evaluate Search Engine

In order to answer to the previous question we went through a four steps research process (see Figure 7). The first step was aimed at defining the right sample to be



**Fig. 7.** The research process

investigated. As discussed in the next section, there are thousands of search engines and the industry is characterized by a heavy concentration index (just one company has more than 80% of the total market). In such an environment the sample selection will be significant as far as it includes the market leader. For this reason, more than looking for statistical significance, by increasing the sample numerousness, we tried to increase the internal and external validity by increasing the sample variance, as it is described in the Section 2.1.

Once defined the research sample, we conducted some preliminary case analysis, based on secondary data, and used the empirical observation to draw a simplified model able to describe the service architecture adopted by different players. In the same time the model was aimed at becoming the data-gathering tool for the extended research, thus, considerable effort was put to clearly define the different evaluation grids for each variable of the model, as it is described in the Section 2.2.

Finally the model and the evaluation grids were used to map all the 26 cases of the sample. The gathered data allowed to conduct an analysis of positioning identifying different strategies coexisting on the market, ranging from companies that are clearly betting on a specific part of the service architecture, to companies trying to excel on all of them. The results of the data gathering and of the following analyses are described in Section 3.

## 2.1 Sampling

The research process started with the selection of a sample of web search engines, used in the following steps to create and validate the model. The universe of search engines is difficult to evaluate, because no registry of them exists. Blogs and specialized web sites<sup>2</sup> give an esteem of their number that varies between 1500 and 2000.

The sample analyzed consists of 26 search engines, which represent about 2 or 3% of the whole universe. This could be seen as reductive, but recent data from market-share.hitslink.com show that the share of searches is concentrated on few search engines (Google, for example, attracts about the 81% of searches, followed by Yahoo!, with 7%).

<sup>2</sup> See, for instance,

<http://www.boutell.com/newfaq/misc/howmanysearch.html>

Starting from the above consideration the objective in the sample selection was twofold. On the one hand, in order to increase the external validity of the research, we decided to consider in the sample different categories of search engines. In particular we focused on general web search tools (e.g. Google); content related search tools (e.g. Midomi); search tools that aggregate results from other major search engines (e.g. Leapfish) and “innovative” search services difficult to be categorized that are probably opening new categories (e.g. Wolfram Alpha).

Moreover, in order to increase the internal validity, we also decided to consider more than one case for each category. The final sample is thus described in Table 1.

**Table 1.** The search engine we considered in our sampling

Category	Search Engine	
General Web search tools	Google [11] Yahoo! [12] Bing [13] Cuil [27]	Ask [14] Gigablast [40] Exalead [44]
Content-related search tools	Music - Midomi [15] Images - RETRIEVr [16] - TinEye [17] Real Time Information - OneRiot [22] - Yauba [43]	News - Daylife [18] People - ZoomInfo [19] - Wink [20] Question&Answers - Answers.com [21] - ChaCha [42]
Meta search engine	Leapfish [25] WebCrawler [26]	Zuula [28] Copernic (formerly Mamma) [45]
Innovative search service	Powerset [41] Kosmix [23] Wolfram Alpha [24]	Evri [29] TrueKnowledge [46]

## 2.2 A Search Engine Model Definition

In order to perform a comparative evaluation of the various search engines, we had to define a search engine model with two characteristics:

1. *simple* enough to serve as a agile evaluation tool and
2. *general* enough to describe a wide spectrum of search engine.

For these reasons, we decided to opt for a process model that covers the various activities the users and the search engines perform (see Figure 8).

The process begins with users looking for something in the rich offering of the web. The most straight forward step is directly submitting their *requests* to the search engine, however search engines often offer a variety of alternative actions to be perform before submitting the request. Several search engines, for instance, offer users the possibility to restrict the search space by choosing a category. Others offer to get personalized results if the users log-in. We collectively aggregate all these offerings in the *pre-filtering* step.

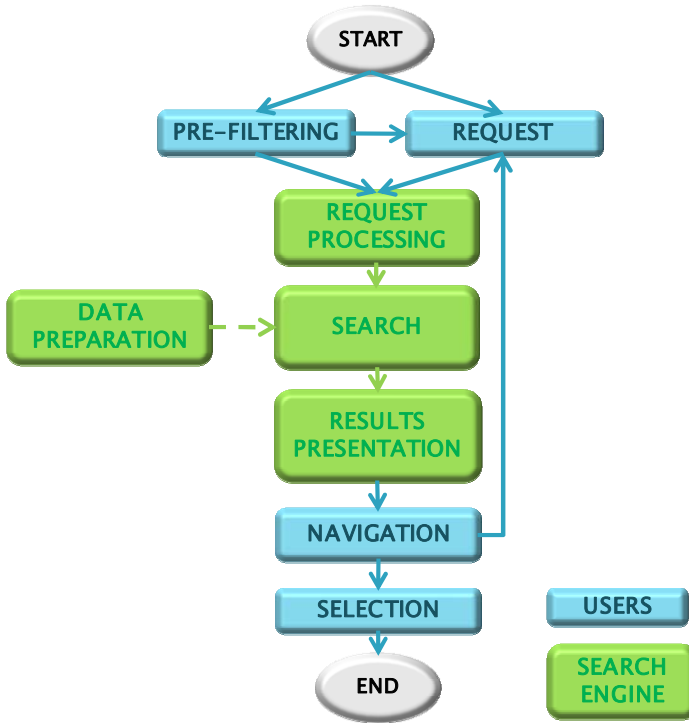


Fig. 8. The activities that occur every time a user makes use of a search engine

At this point search engines start their work. A first internal step is *processing request*; this can be as simple as breaking the search phrase in key words or as complex as using natural language processing techniques to guess the user intention. A second step is *searching*. When considering a Web search engine like Google or Yahoo!, this step is realized by matching the keywords against the indexes constructed off-line by crawling web pages. When considering a meta search engine like WebCrawler, this step is realized by routing the request towards a set of external Web search engines, that will answer the user request in parallel. In both cases an off-line step, which we named *search preparation*, is required either (a) to crawl the Web, elaborate the crawled content (e.g., annotation step in multimedia search engines) and construct indexes, or (b) to select external search engines to route the query to. A final internal step consists in *presenting the results* to the user. This can vary from compiling a list, as done by most search engines, to apply sophisticated techniques to aggregate ranking from multiple search engines, as done by WebCrawler.

Then is time for the user to *navigate* the result set and *select* some of the results. In case of negative results, *the process can be iterated* several times and eventually the user leaves the search engine web site.

## 2.3 A Simple Evaluation Method

This model is the result of several iterations over model definition and model testing on our sample of search engine. In each of the iterations, we identified a set of drivers in each step of the model and we evaluate the offering of each search engine.

In order to solicit each search engine in a comparable way we identified the following simple but challenging set of requests (they are ordered by increasing complexity):

- looking for "activities" (plural) and getting results also matching "activity" (singular) – it requires the search engine to apply stemming techniques;
- looking for "jaguar" and getting results separated in categories such as animal, car, band, video games, football team, etc. – it requires the search engine either to hold a taxonomy of the alternative meanings or to cluster results;
- looking for "Milano" and getting results separated in categories such as city, actor, etc. – it is similar to the previous one but it introduces a language flavor in the processing being Milano the Italian name of Milan and a surname of an actress;
- looking for "Milan Berlin" and getting travel results first – it requires the search engine to guess the possible user intention and act correspondingly;
- looking for "Paris Hilton" and getting news and photo of the VIP vs. looking for "Hilton Paris" and getting results about the hotels in Paris – it is similar to the previous request, but it requires much more smart techniques to be put in practice on a large scale.

In the case of domain or media specific search engines, for which these requests do not always apply, we made use of requests of similar complexity that such search engines were able to process.

The content of the following sections is twofold. In each section, we first describe the final set of drivers identified for each step in the model. And then, we describe how we were able to evaluate each search engine by rating how much it bets on the various steps. Ratings vary between 1 star (★) to 3 stars (★★★), where one star represents a poor presence of the considered dimension, while 3 stars are usually linked to those driver hold as strengths of the search engine.

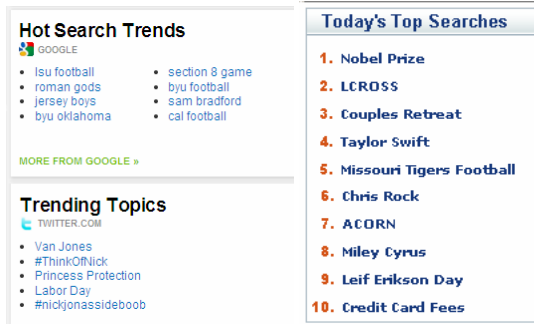
### 2.3.1 Pre-filtering Step and Its Evaluation Criteria

A search engine can provide users with pre-filtering tools that give them the possibility to get to the desired results without entering any word or with little effort. The pre-filtering tools we found in our sample are: topic sections, search trends, recent answer, automatic suggestions, personalization functions, search memory, homepage customization, and search options.

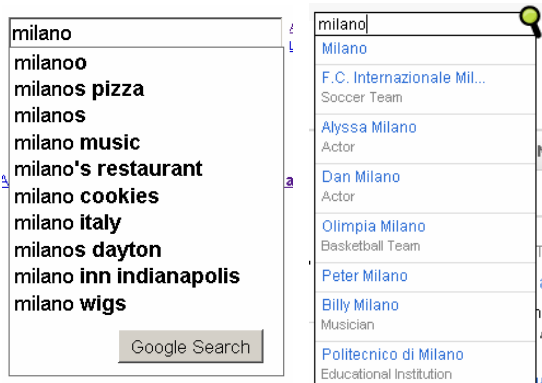
**Topic Sections** are sections dedicated to specific themes or topics (health, shopping, sport, politics, etc.) in which users can find suggestions useful to reach what they're looking for before starting the research process. Topic sections are usually settled in the home page, so they also contribute to the attractiveness of the search engine. A good example, see Figura 9.a, is provided by Evri where a box with links on themes regarding politics, business, entertainment and sport is accessible from the home page.



(a) Example of Topic Sections from Evri



(b) Example of Search Trends from KOSMIX (left) and Yahoo! (right)



(c) Example of Automatic Suggestions from Google (left) and Evri (right)

**Fig. 9.** A set of screen-shot illustrating some of the pre-filtering offering

**Search Trends/Recent Answers** are helpful suggestions provided with the aim of guiding users with popular queries and Q&A. They usually show up in the home page and are typical of social, news and real-time search engines. A good example, see Figura 9.b, is provided by KOSMIX where the sections - called Hot Search Trends or Trending Topics - can be considered examples of these functionalities and they are both present in the home page. Also Yahoo! has a similar box named Today's top searches.

**Automatic Suggestions** are keywords or categories automatically suggested to users during data input. These suggestions usually appear in a dropdown menu under the input field. Automatic suggestions can be based on popular and recent searches – this is the case of Google shown in Figura 9.c on the left – or on a result of the data preparation process – this is the case of Evri shown on Figura 9.c on the right.

**Personalization functions** provide users with specific suggestions after a log-in to the web site of the search engine (this is possible thanks to implicit and explicit profiling of the same users). For example, results can be shown or ordered in a different way referring to the user’s search history or it is possible to comment results and to mark them in order to easily recover them in the future. A good example is provided by the i-google service.

**Search Memory** is a tool that gives the user the possibility to access easily to his past searches and eventually recall them. Every input is stored and available for a certain period of time. A good example is provided by ZOOA where *Recent Searches* box shows all the recent words typed in the search box.

**Homepage Customization** possibility for the user to create his own customized home-page by choosing different boxes and sections but also display a different layout and activate different tools. Also this kind of customization is part of the new “I-google” where users can create their own homepage after logging-in the web site.

**Search Options** are functions and filters available for the user in order to formulate more detailed and specific queries. Search options can be found in the home page, usually near the input search bar, or in a dedicated section called *advanced search*. In the first case, options are evident and can be used easily, while filters available in the advanced search are more likely to be used only for a really specific query. From our sample we have identified the filters available are the following: Geographic, Language, Output type, Boolean logic, Date, Keyword position, Domain (com, it, org, net, etc.), Copyright, Safety option, Similar pages, Links, Number of displayed results/pages, Topic (meta-search), Q&A state (social search), Relevance and Personal data (people search).

In order to evaluate how much each search engine is betting on the pre-filtering step, we considered the presence of the tools described above with the exclusion of the search options and we created four categories:

- A. only basic automatic suggestion is available;
- B. one tool, but the automatic suggestion, is available;
- C. 2 or 3 tools are available; and
- D. more than 3 tools are available.

For search options, we created an index  $\alpha$  whose values can be 0 if no search option is present, 1 if one or two search options are present and 2 if more than three options are present. We assigned the stars using the table illustrated in Table 2.

**Table 2.** The method used in assessing how much the search engines bet on the pre-filtering step

		Pre-Filtering Categories			
		A	B	C	D
Search Options Index	$\alpha = 0$	★	★	★★	★★
	$\alpha = 1$	★	★	★★	★★★
	$\alpha = 2$	★★	★★	★★★	★★★★



### 2.3.2 Request Step and Its Evaluation Criteria

This step highlights *how* the user makes clear what he's looking for with regard to the type of input allowed in the search engine. This is an important step in the research process and it is strictly connected to the efficacy of pre-filtering tools, because the more useful are those tools the less used will be this functionality.

In our sample we detected the possibility to submit different media types:

- **Text** in the form of single or multiple keywords, phrases or even complete questions in natural language;
- **Image** in the form of a sketch to be drawn in ad-hoc spaces (this particular input tool can be found in RETRIEVr, a search engine that has access to the vast amount of pictures uploaded on Flickr.com) or a URL of an image present on the Web (this is the case of TinEye).
- **Audio** in the form of sounds recorded, singed or hummed by the user (as in Midomi).

In evaluating how much the various search engine bet on this step we assigned the starts as follows:

- ★ means that only textual input is allowed;
- ★★ means that image or audio input is allowed; and
- ★★★ means that more than one kind of input is allowed.

### 2.3.3 Request Processing Step and Its Evaluation Criteria

Under the term *query processing*, we group all those technical activities that are put in place before executing the actual search. These activities can range from simple ones, such as stemming to the attempt to detect user intention.

An example of such an attempt is provided by Google. Google interprets differently the search for “Paris Hilton” and the one for “Hilton Paris” (see Fig. 10). More advanced techniques, such as Natural Language Processing, are deployed in several Google's competitors. For instance, asking “What is the temperature in Boston?” to Wolfram Alpha results in the most recent temperature measured in Boston (Massachusetts, USA), but the system also asks whether the user means Boston (UK) or other four towns named Boston around the world.

The figure displays two side-by-side screenshots of Google search results. The left screenshot shows a search for "paris hilton" with results for news articles about her. The right screenshot shows a search for "hilton paris" with a sponsored link for "Hilton Hotel Paris" and a map of the hotel's location in Paris.

**Fig. 10.** The two screenshots above shows the ability of Google to distinguish the user intention. On the left the user searches for “Paris Hilton” and Google first answers are news about the VIP. On the right, the user searches for “Hilton Paris” and Google first answer is the position of the hotel in Paris.

In evaluating how much the various search engine bet on this step we assigned the stars as follows:

- ★ means that the analyzed search engine does nothing special, for instance it only applies stop words;
- ★★ means that it applies well known techniques such as stemming to avoid singular/plural mismatch; and
- ★★★ means that it applies experimental techniques such as NLP, computation of audio-fingerprint, real time multimedia feature extraction, intention detection.

### 2.3.4 Data Preparation Step and Its Evaluation Criteria

Under the term *data preparation* we group several technical activities. For google-like search engine data preparation is not performed at query time and indicates: crawling, analyzing, indexing, and ranking resources as described in [32]. Search engines tend to differentiate themselves on the basis of which activity they stress more. For instance, Google has been stressing the usage of PageRank, while CUIL, on the contrary, puts lot of emphasis on crawling. On the contrary, for meta search engine, data preparation consist in selecting the search engines or the data source to ask at run time. Other approaches are also possible; for instance Kosmix approaches to the Deep Web [39] leveraging a huge taxonomy of millions of topics and their relationships.

In evaluating how much the various search engines bet on this step we assigned the stars with two different criteria, one for the search engines (e.g., Google, Yahoo!, Midomi, RETRIEVr) and one for the meta-search engines. For the former the assignment follows the rules listed hereafter:

- ★ means that the analyzed search engine applies standard crawling and indexing techniques such as those described in [32];
- ★★ means that it applies the previous techniques and other advance once such as PageRank, acoustic-fingerprints [37] and multimedia feature extraction [38];
- ★★★ means that it applies the standard and the advance techniques together with some more experimental approached such as latent semantic indexing, NLP, ontologies and taxonomies.

For the meta-search engine we used the following rules instead:

- ★ means that it invokes few well know search engines (e.g., Google, Yahoo! and Ask) or it access few well know web sites (e.g. LinkedIn, Facebook, Fliker and YouTube);
- ★★ means that it selectsthe external search engine to invoke or the external web site to access depends on the actual query submitted by the user;
- ★★★ means that not only it cleverly selects external services and web sites, but it combines such information with self crawled web pages or access to a wide range of Deep Web sources.

### 2.3.5 Search Step and Its Evaluation Criteria

The search step occurs just after the request processing and operates over the data and services prepared by the search preparation step. When considering a Web search engine like Google or Yahoo!, this step is realized by matching the keywords against the indexes constructed off-line by crawling web pages. When considering a meta search engine like WebCrawler, this step is realized by routing the request towards a set of external Web search engines, that will answer the user request in parallel. As we will discuss in the conclusion, this step is particularly relevant for Kosmix and Wolfram Alpha.

As in the case previously described of evaluating the search preparation, we assigned the stars with two different criteria, one for the search engines (e.g., Google, Yahoo!, Midomi, RETRIEVR) and one for the meta-search engines. The rules for the search engines are the following:

- ★ means that the analyzed search engine limits the search to keyword or multimedia feature matching;
- ★★ means that it applies well know techniques of the previous case, but it also computes facets, clusters, related search and other similar features that depend on the query issued by the user; and
- ★★★ means that, in addition to the two previous techniques, it applies also some experimental techniques such as suggesting alternative interpretation of the meaning associated to the user request.

For the meta-search engines we used the following rules instead:

- ★ means that it limits the search to broadcasting the same request to all selected search services and data sources;
- ★★ means that it does not broadcast the same request to all the selected search services, but it adapts the request to the peculiarity of the search service/data source. In addition to this it also combines the results, e.g., by applying rank aggregation; and
- ★★★ means that, in addition to the previous techniques, it tries to aggregate the retrieved information in an unified view.

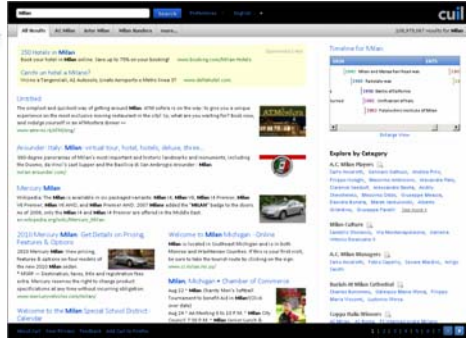
### 2.3.6 Results Presentation Step and Its Evaluation Criteria

This step is strictly related to the previous one: it presents its results. Screenshots of four different result pages from Google, Cuil, Kosmix, and Wolfram Alpha are show in Fig. 11.

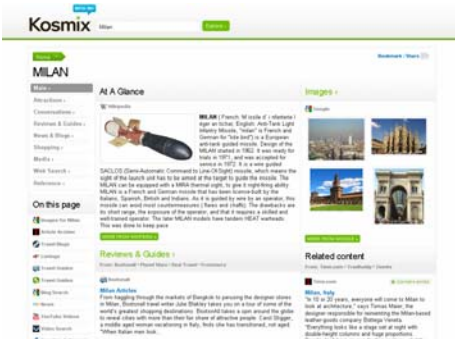
Google presents list of results; having detected that the user is asking for the Lombardy City it puts a map and some photos among the results. Cuil proposed a page mixing result listing in Google-style with info boxes as it was a magazine. Cuil also proposes tabs with alternative meanings of the keyword “Milan”; it proposes AC Milan, Inter Milan, Milan Cundera, and some ten more. Kosmix opts for the magazine style with several boxes presenting articles about the possible interpretation of the keyword “Milan”: a Wikipedia article about a French missile named Milan, weather reports in Milan, shopping places, and so forth. Finally Wolfram Alpha reports facts it knows about Milan such as the population, the geo-coordinates and the current weather. Notably it tells the user that it was assuming "Milan" is a city but it can use it as a given name.



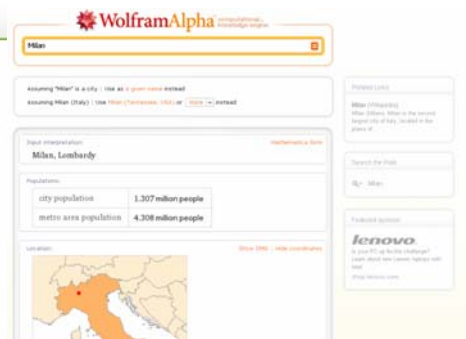
(a) Google



(b) cuil



(d) Kosmix



(d) WolframAlpha

Fig. 11. The different results presentation of the query “Milan”

To evaluate how the different search engines bet on this step we compiled the following list of rules:

- ★ means that the analyzed search engine does nothing special, e.g. it presents result in a plain list such as Google;
- ★★ means that it presents of facets, clusters, filtering options, related searches, reordering button, page thumbnails and similar features; and
- ★★★ means that it tries to organize the content as if it was prepared by a person, e.g. the magazine style approach of Kosmix.

### 2.3.7 Navigation Step and Its Evaluation Criteria

The last step is the navigation one. It encompasses actions for refinement and manipulation of output data. These actions are similar to the navigation of standard website, but are usually not desirable when dealing with search engines because users want to get to useful results as soon as possible.

However, most search engines offer several post-search filtering options, which can be re-arranged and customized on personal needs. These options partially overlap with search options presented in Section 2.4 when discussing the pre-filtering step.

Displayed results can normally be arranged (or clustered) along one or more of the following dimensions: file size, date, relevance, format, topic, language, personal data (clustering of people by age, job etc., typical of *people search*), results from a specific page (this option is available, for example, for registered users on *Google* and it's a default option on *Gigablast*), search engine used (this last option is typical of meta-search that relies on external search engines).

We can also find other options such as:

- connections – it offers the user list of links (or even diagrams in the case of Evri) to related topics which the user may be consider of interest;
- translation service – it allows users to access in their language pages written in foreign languages;
- results sharing – it offers the possibility to share the results of queries on social networks; and
- social networking – it allows registered users to interact with other users interest to the same topics.

In order to evaluate how much each search engine is betting on the navigation step, adopt the following strategy, we give ★ if zero or one option is present, ★★ if two or three options are present and ★★★ if more than three options are present.

### 3 Results

According to the previous model we mapped all the 26 search engines of the sample. Even though the evaluation grid was developed to make the evaluation as objective as possible, we decided to have each single site assessed by four different researchers and then we compared the results converging to the final evaluation table below (see Table 3). It is important to underline that the score given to each item is an absolute measure (according to the evaluation grid) and not a relative one. In other words the three stars on the Navigation item of Google and OneRiot mean that they show comparable features on this dimension.

The data gathered allowed to conduct an analysis of positioning and to identify the presence of different strategies coexisting at the same time on the market. First of all a Factor analysis was conducted to reduce the number of items allowing to reduce the 7 starting items into 3<sup>3</sup>.

- **Factor 1: Search Preparation**

This factor is composed by Request Processing and Data Preparation. It represents the effort in preparing the database to be searched.

- **Factor 2: Search**

This factor is composed by Search and Result Presentation. It represents the effort in preparing searching the database and proposing the results to the users.

---

<sup>3</sup> Extraction method: principal component analysis; rotation method: varimax with Kaiser normalization; all the factor loading are higher than .5 and the Crombach's Alpha is higher than .5.

**Table 3.** Results of the mapping process

SEARCH ENGINE	PRE-FILTERING	REQUEST	DATA PREPARATION	REQUEST PROCESSING	SEARCH	RESULTS PRESENTATION	NAVIGATION
GOOGLE	***	*	**	***	**	*	***
YAHOO!	***	*	**	***	**	**	**
BING	***	*	**	**	**	**	***
CUIL	*	*	***	**	**	**	**
ASK	**	*	**	**	*	*	*
GIGABLAST	*	*	**	*	*	**	**
EXALEAD	***	**	**	**	*	**	**
MIDOMI	**	***	***	***	*	**	**
RETRIEVr	*	**	***	***	*	*	*
TINEYE	*	**	***	***	*	*	**
ONERIOT	**	*	*	*	*	*	***
YAUBA	*	*	**	**	***	**	*
DAYLIFE	**	*	***	**	*	***	**
ZOOMINFO	*	*	**	*	*	**	**
WINK	**	*	**	*	*	**	*
ANSWERS	***	*	**	*	**	**	**
CHACHA	**	*	*	*	*	*	**
LEAPFISH	**	*	**	**	*	**	**
WEBCRAWLER	**	*	**	*	**	*	*
COPERNIC	*	*	**	*	**	*	*
ZUULA	*	*	*	*	*	*	**
POWERSET	*	*	**	***	***	**	*
KOSMIX	***	*	***	**	**	**	**
WOLPHRAM ALPHA	*	*	***	***	***	**	*
TRUE KNOWLEDGE	*	*	***	***	***	**	*
EVRI	**	*	***	**	***	**	**

#### • Factor 3: Pre-Post Search

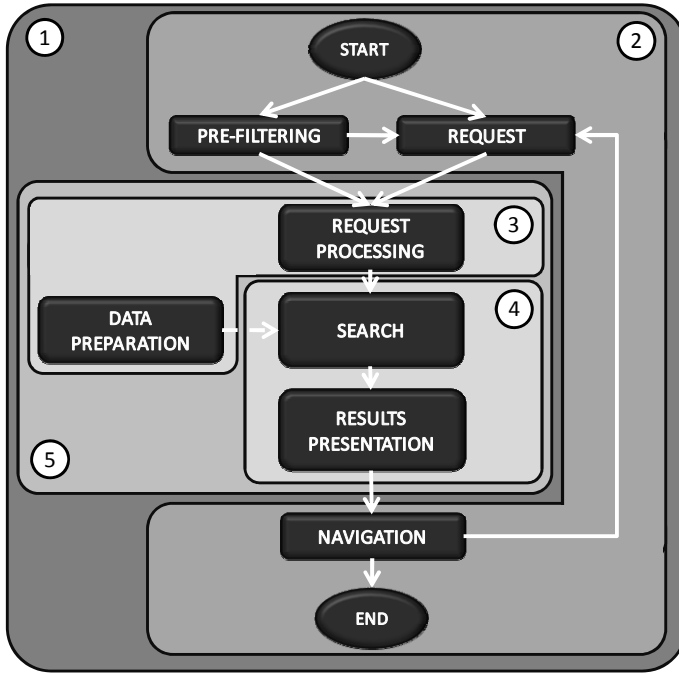
This factor is composed by Pre-filtering, Request and Navigation. It represents extent to which the user can give a contribution before or after the searching activity.

Once identified the three factors, the value of each search engine on these factors was calculated as the average values of the starting items.

Afterwards we wanted to identify what search engines are showing a focused strategy, which means they are putting more attention on one factor. Our question was if there are some companies betting more on one dimension than others. For example it is easy to see in Table 3 that the image search by similarity of TinEye is focusing its effort on Search Preparation, while Google seems to have a more balanced approach showing similar efforts in developing all the three factors. In order to identify what companies are focusing their strategies (and on what of the factor(s)) we created a new binary variable called **focused strategy**. For each search engine this variable equals to 1 if the variance of the scores of the three factors is higher than the average variance in the sample (and it equals to zero otherwise). A post hoc analysis on allowed to identify the following strategies.

1. **Bet on everything<sup>4</sup>**: Google, Yahoo, Bing and Exalead
2. **Bet on user interaction (pre and post search)**: Wink
3. **Bet on preparing for searching**: RETRIEVr, CUIL, TinEye and Midomi

<sup>4</sup> In this category we may have both companies showing significant efforts on all the variables and companies showing poor efforts on all the variables. The companies included in the second category will not be considered from now on.



**Fig. 12.** The five strategies the search engines are following

4. **Bet on searching power:** Evri and Yauba
5. **Bet on searching excellence:** Kosmix, Wolfram Alpha, TrueKnowledge and Powerset

In the following we explain the five strategies in details and we refer to Fig. 12 for visualizing a mapping between the five strategies and the search process we introduced in Section 2.2.

### 3.1 Strategy 1: Bet on Everything

The big names in the search engines industry, i.e., Google, Yahoo! and Bing all appear to be betting on everything.

Google and Yahoo! offer powerful pre-filtering support with iGoogle and MyYahoo!. In a different way, Bing Visual Search<sup>5</sup>, with the possibility to visually select topics to search, is also an outstanding pre-filtering interface.

Considering the request step, they all stick with the standard solution of offering a text field in which user can enter the request, but they apply a wide range of request processing techniques. All three of them support stemming and, in different way they, try to interpret the users' queries to get their intentions (example were presented in Section 2.6).

<sup>5</sup> <http://www.bing.com/visualsearch>

	Google		Yahoo!		Bing	
	# results	support	# results	support	# results	support
"Activities" - it requires stemming	329	yes	1860	yes	232	yes
"jaguar" - it requires taxonomies or clustering	46	no	300	somehow	63	almost
"Milano" - it requires taxonomies or clustering and language detection	95	no	409	no	62	almost
"Milan Berlin" - it requires user intention detection	14	yes	65	somehow	14	somehow
"Paris Hilton" vs. "Hilton Paris" - it requires intention detection on a large scale	46-21	yes	202	no	50	no

**Fig. 13.** A comparison between Google, Yahoo! and Bing on the evaluation queries proposed in Section 2.3

It's difficult to judge whether the three major search engines differ much in data preparation, search, and result presentation steps. Using the evaluation queries presented in Section 2.3 we got the results shown in Fig. 13. Yahoo! outperforms both Google and Bing in terms of number of results; it gives five times more results. This may indicate that Yahoo! bets a bit more on data preparation. However, Bing and Yahoo! handle almost correctly the query on Jaguar and Milano. Both of them propose a "related search" or a "search refinement" box that helps in overcoming the ambiguities in the query. In this Bing performs slightly better than Yahoo!, for instance, Yahoo! searching for Jaguar retrieves only the car vendor related results and proposes links to search for specific Jaguar models. Bing, on the contrary, proposes a mix of results and proposes to search for "Jaguar Animal", "Jaguars Stadium", and some ten more options. Finally, all three search engines handle the Milan-Berlin query proposing travel related information first. The only search engine that manages the "Paris Hilton" vs. "Hilton Paris" query correctly is Google (for more info see discussion in Section 2.6).

Moreover, all three search engines have several experimental features that explore almost all the steps. Just to cite few of them:

- Yahoo! has been spending a lot of effort on SearchMonkey<sup>6</sup>, a developer tool to extract structured data from the Web (i.e., microformat and RDFa) and build applications to display custom enhanced results.
- Google recently proposed Fusion Tables – a service for managing large collections of tabular data in the cloud –, Google Squared – a service that fetches facts from the Web and organizes them – and Flu Trends – an experiment in calculating up-to-date estimates of flu activity using search trends.

### 3.2 Strategy 2: Bet on User Interaction (Pre and Post Search)

Search engines betting on this strategy invested in state-of-the-art technologies for data preparation, request processing and search, but they excel in pre-filtering options and result navigation.

Wink is an emblematic case of search engine that bets on user interaction. The home page of the people search engine allows users to take several pre-filtering actions. Users can tell Wink the intention to search for different kinds of people (e.g., friends, classmate, co-workers) and different types of information (e.g., emails, phone numbers, online presence in social networks). In Fig. 14, we present a screen shot of the pre-filtering options offered by Wink.

<sup>6</sup> <http://developer.yahoo.com/searchmonkey/>



**Email Search:** [Free People Search](#) | [UK People search](#) | [Canada People Search](#)  
**People Finder:** [Find Friends](#) | [Find Classmates](#) | [Find Co-workers](#) | [Advanced Search](#)  
**Background Check:** [Background Check](#) | [Detailed Criminal Records Check](#) | [Public Records Check](#)  
**White Pages:** [Phone Number Search](#)  
**Find a Person:** [Find a person online](#) | [Names and Screen Names](#) | [Social Network Search](#)

---

[About](#) ▪ [Blog](#) ▪ [Developer](#) ▪ [Forums](#) ▪ [Feedback](#) ▪ [Privacy](#) ▪ [Terms](#) ▪ [Jobs](#) ▪ [Directory](#) ▪ [Names](#) ▪ [Tools](#) ▪ [In Ti](#)  
 Copyright © 2007 Wink Technologies, Inc. All Rights Reserved.

**Fig. 14.** The pre-filtering options offered by Wink

The results of a search are presented as a list, but each result links to a special page where all the information about the found person is displayed in a profile. Such a profile looks like the personal profile a person can manually create on a Social Network, but it is automatically created with the information found on the Web. Navigation tools are offered to explore the profile and to isolate the information the Wink user is looking for.

### 3.3 Strategy 3: Bet on Preparing for Searching

RETRIEVr, TinEye, Midomi and Cuil are very good examples of search engines whose strategy is to bet on preparing for searching.

RETRIEVr and TinEye are image search engines; they both excel in finding photos on the Web. To do so, they extensively extract multimedia features of images published on the Web (i.e., betting on data preparation) and index them. Moreover, they both offer the possibility to point to an image on the Web as a way to indicate what the user is searching for. RETRIEVr also offers a sketch pad where users can draw what they are looking for. The request processing step consists in extracting multimedia features from the users' input using the same techniques employed in the data-preparation phase and in matching them against the indexes.

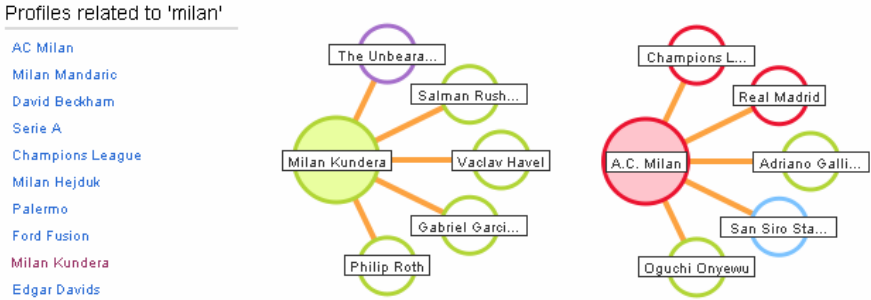
Midomi operates in a similar way, but it concentrates on music. It applies the innovative techniques of acoustic fingerprints to songs and users' input. Users are invited to sing or hum a song for 10 seconds and Midomi finds songs in its archive that match the users' input.

Cuil, differently from the formers, does not bet on image and song feature extraction. However, it reached worldwide visibility for having claimed to have crawled more Web pages than any other Web search engine. It clearly bets on data preparation as a key ingredient of its strategy.

### 3.4 Strategy 4: Bet on Searching Power

This fourth strategy is complementary to the previous one. Search engines, which bet on this strategy, are outstanding in search and result presentation. To do so, they also have to excel on data preparation, but they don't bet much on request processing. Evri and Yauba are two emblematic cases of bet on search power.

Evri tries to replace the need for search engines by suggesting that users explore how entities are connected to each other. Whenever a user searches on Evri, it gets the



**Fig. 15.** Two distinguishing features of the Evri search engine: a related profiles box (on the left) and two diagrams showing an entity and its connections (on the right)

list of results found on the Web, but Evri also displays a box which it populates with known entities and links to Evri’s database. These links point to hundreds of thousands of people, products and things. In Figure 15, we show on the left an example of the links listed in such a box when searching for “Milan” and on the right two examples of the entities the users can explore.

Yauba is a different example of betting on search power. It seeks to transform the way people find information online, while providing maximum protection for their safety, security and privacy. Yauba does not bet on the size of the index<sup>7</sup>, it bets on intelligence. Yauba tries to tell the difference between concepts such as “Milano” the Italian city, “Milano” the Texas city, “Milano” the actress (and much more). It does so using word-sense disambiguation extraction technologies. Moreover, Yauba completely rejects the view that search engines should keep mountains of data on their own users. Instead, it takes the exact opposite approach. It protects the privacy of its users. It keeps no record of search terms, browsing habits or any other personally identifiable information.

### 3.5 Strategy 5: Bet on Searching Excellence

The search engines, which bet on searching excellence, concentrate their attention to the four inner blocks of our model: request processing, data preparation, search and result presentation. Kosmix, Wolfram Alpha, TrueKnowledge and Powerset appear to be those that adopt this strategy successfully. They differ a lot in terms of results, but they clearly excel in the four inner blocks. Hereafter we compare the answers of the four search engines when we ask them the four evaluation queries presented in Section 2.3.

All of them are able to process the query “activities” that requires stemming. The answers of the four search engines differ a lot. Kosmix constructs a magazine-style page in which it collects bits of the Web from multiple sources about the concept “activity”. Wolfram Alpha tells the user the six meanings of the noun “activity”, its pronunciation, its synonyms, and a set of narrower and broader terms. TrueKnowledge answers that “activity” is an “intransitive action” and similarly to Wolfram

<sup>7</sup> Yauba About page states that: “Most search engines like to brag about the size of their index. At Yauba, we could not care less. What we care about is *intelligence*.”

Alpha it shows links to narrower and broader terms. Powerset presents a set of Wikipedia pages that match “activities” or “activity” and then tells the user that it knows several type of activities (e.g., ski, hike, metallurgy, drama, etc.) and that activities lead to sounding, award, expansion, demands, and unification.

When answering the second evaluation query – jaguar – Kosmix presents a magazine-style page describing the animal, but it asks whether the user is interested in Jaguar the car or Jaguar the Atari console, etc. Wolfram Alpha distinguishes between Jaguar as an English word and Jaguar as a species specification and answers accordingly. TrueKnowledge shows the different possible meanings in separated boxes and it lists a set of super and sub classes for each meaning. For instance, for Jaguar as cars it lists all the 40 different Jaguar models it knows. Powerset acts similarly distinguishing between seven possible meanings: the animal, the car vendor, the car models, the band, the Marvel comics, the rocket and the chemistry software. For each of them it presents a list of relevant Wikipedia pages.

We obtain a similar result when we evaluate the third and the fifth query: Milan and “Paris Hilton” vs. “Hilton Paris”. The four search engines are able to pre-process the request and get the possible meanings or intentions, search for one or more of these meanings/intentions and prepare a presentation of the results that helps the user in understanding that the request was ambiguous and different results are available for different interpretations. The four search engines differ in the number of meanings and intentions they know and in the presentation of the results.

A query that seriously challenges all the four search engines is the “Milan Berlin” one. As we explained in Section 2.3, this query requires the search engine to guess the possible user intention (i.e., getting travel results first) and act correspondingly. Kosmix, TrueKnowledge, and Powerset do not detect the intention. The results of Kosmix and Powerset contain contents from Milan and Berlin, while TrueKnowledge answers that it does not understand the question. Most likely this behavior is a result of the design decision to realize a search engine that constructs a result set around a given topic. As a result Kosmix, TrueKnowledge, and Powerset do not handle multi-topic queries. Differently from the others, Wolfram Alpha detects the user intention and provides information such as the distance between the two cities and the path an airplane should follow, but it does not point to any relevant service that can help the user in arranging the travel.

## 4 Discussion

This research showed that something in the search engine industry is moving. There are many clues that seem to forerun the end of a static phase and the opening of a new fluid one.

First of all we can observe that in the last years the number of new players on the market increased. These players are far from creating market problems to Google, but it is a fact that large companies like Microsoft (recently with Bing) and new ventures like Wolfram Alpha are looking for new ways to answer to the market needs. Moreover the existence of Search Computing (SeCo) and the university research is converging on the same market making it clear that something could happen in a short time. We can also observe that this growth in the number of players is often pushed

by new technologies (e.g. vocal recognition or large images dataset). Still it is interesting to observe that also already existing (perhaps disruptive) technologies and solutions like the metasearch (Webcrawler or Copernic) are receiving new blood.

In other words our analysis of positioning allowed to see that some players are stressing the current dominant architecture by investing heavily on new technologies. In some cases their effort for offering something new to the customers is not limited to some specific components but it implies re-discussing the whole service architecture going back in the design hierarchies [9] (e.g. the Kosmix metasearch architecture that is different from the current dominant design proposed by Google). In other cases their effort still fit inside the dominant architecture (e.g. the innovative techniques of acoustic fingerprints by Midomi), but is not hard to think that the growth of radically new components will challenge the existing architecture asking for a new one.

Finally we can observe that a clear new way hasn't emerged yet. The present research shows that many companies are seriously challenging the current dominant design and that a large set of alternatives is offered to the market, that are clear signals of a market entering in a new fluid phase [2].

However, these considerations do not imply that we will have a new market leader soon! Many researchers showed that in rapid moving environments the market leaders developed the capability to react to new entrants and to maintain their leading position [10]. This means that it wouldn't be surprising if the company introducing the new Dominant Design will be Google again.

Thus this research allows to raise some specific questions which are relevant to the SeCo project concerning the next Dominant Design.:

- When will it emerge?
- Which technologies will it be based upon?
- Will this be coherent with SeCo's vision?

Even though answering to these questions is a difficult and major task, this is not the only one. The present research focused mainly on the service offer to the market. Still, the analysis of the service model should be accompanied by the analysis on the business model as well. The current service dominant design is connected to a dominant business model that mainly leverages on the sponsored links. Still, there are many other possible revenue streams that can be exploited (e.g. charging users for searches or technology licensing). A short description of the most acknowledged ones is listed in Table 4.

It is interesting to observe that some of the players that are proposing focused strategies are not leveraging on sponsored links as main revenue stream. For example Evri, Exalead, and Copernic seems to bet on selling technologies/technology licensing, while Midomi seems to bet on selling contents to the final users.

These considerations are worth of a deeper investigation but are already enough to open some preliminary questions regarding the SeCo business model:

- Who are the SeCo customers? Final users or search companies?
- Is it better to have a close or an open approach to SeCo trading?
- How can we adapt the existing revenue sources to a multi-layer architecture?

**Table 4.** Main sources of revenues

**Sponsored links:** some search engines sell keywords. Advertisers buy keywords to come up first when the corresponding query is submitted. Selling keywords is the most popular and effective way of making revenues for search engines.

**Banners:** this spread moneymaking technique, which belongs to that marketing area called marketing promotion online, is based on direct advertising. Several search engines, as web pages in general, host advertisement (called banners) with the aim of attracting users to the advertisers' web pages.

**Charging users for searches:** Another possibility to earn money is to make users pay a fee for the access to the search service. The difficulty with such a technique is that there is a limited base of users that want to pay for search tools, so this strategy is less attractive for advertisers. Charging users is actually limited to those special value-added searches or to access relevant articles that generally provide information of higher quality in comparison to the results displayed by common search engines.

**Free search and contents with fee:** in this case, the use of search engines is free-of-charge but all the contents on the web site (songs, videos, documents, etc.) are accessible only by paying a fee. Search engines help customers in finding the content they intend to buy and this service is for free.

**Search listing:** sometimes search engines charge advertisers for listing or for a better position in the page of results. A strong limit of this business model, from users' perspective, is that it can jeopardize relevance and pertinence of the results.

**Collection of marketing data:** search engines have the opportunity to collect information about users in different ways. Firstly, it's usually possible to register a personal profile (subscription), which is enriched of information every time the user submits a query. A whole industry has grown up to turn clicks into customization, thanks to the huge quantity of personal data collected. These data can be used to target advertising on specific web sites or they can be sold to marketing agencies. Also search habits are important to turn data into marketing information, in fact every research session helps the search engine in profiling the user and his interests: this is a major ethical issue for what concern users' privacy.

**Technology licensing and software as a service:** some companies that develop search engines can license their search technology to third parts, although this business does not represent usually an important stream of revenue. Search engines can also sell their search service to other web sites or other portals. In this case customers put on the homepage a search tool for contents on their web sites (or for the entire web), the management of which they entrust to the search engine company.

## Acknowledgements

Authors especially thank Silvia Palermo e Claudia Nasuti for the operational support they provided to this research and the fruitful hours of discussion we spent together. This research is funded by the ERC project Search Computing.

## References

1. Utterback, J.M.: Mastering the dynamics of innovation. Harvard Business School Pr. (1996)
2. Tushman, M.L., Anderson, P.C., O'Reilly, C.: Technology cycles, innovation streams, and ambidextrous organizations: organization renewal through innovation streams and strategic change. *Managing strategic innovation and change*, 3–23 (1997)

3. Abernathy, W.J., Utterback, J.M.: Patterns of industrial innovation. *Technology Review* 80(7), 40–47 (1978)
4. Cusumano, M.A., Mylonadis, Y., Rosenbloom, R.S.: Strategic maneuvering and mass-market dynamics: The triumph of VHS over Beta. *The Business History Review* 66(1), 51–94 (1992)
5. Bower, J.L., Christensen, C.M.: Disruptive technologies: catching the wave. *Harvard Business Review* 73, 43 (1995)
6. Gawer, A., Cusumano, M.A.: Platform leadership: How Intel, Microsoft, and Cisco drive industry innovation. Harvard Business School Pr. (2002)
7. Crowell, G.: Search Engine Watch, December 16, 2003 A special report from the Search Engine Strategies 2003 Conference, San Jose, CA, August 18-21 (2003)
8. Yin, R.K.: Case Study Research: Design and Methods. *Applied Social Research Methods Series*, vol. 5, 1(2), p. 3. Sage, Thousand Oaks (1994)
9. Clark, K.B.: The interaction of design hierarchies and market concepts in technological evolution\* 1. *Research Policy* 14(5), 235–251 (1985)
10. Bessant, J., Lamming, R., Noke, H., Phillips, W.: Managing innovation beyond the steady state. *Technovation* 25 (2005)
11. Google Web search service, <http://www.google.com>
12. Yahoo! Web search service, <http://it.yahoo.com/>
13. Bing Web search service, <http://www.bing.com>
14. Ask Web search service, <http://it.ask.com/>
15. Midomi music search service, <http://www.midomi.com/>
16. RETRIEVR flicker image search service,  
<http://labs.systemone.at/retrievr/>
17. TinEye Web image search service, <http://tineye.com/>
18. Daylife news search service, <http://www.daylife.com/>
19. ZoomInfo people and company search service, <http://www.zoominfo.com/>
20. Wink people search service, <http://wink.com/>
21. Answer.com answer search service, <http://www.answers.com/>
22. OneRiot real-time Web search engine, <http://www.oneriot.com/>
23. Kosmix Deep Web search service, <http://www.kosmix.com/>
24. WolframAlpha computational search service, <http://www.wolframalpha.com/>
25. Leapfish Web meta search service, <http://www.leapfish.com/>
26. WebCrawler Web meta search service, <http://www.webcrawler.com/>
27. Cuil Web search service, <http://www.cuil.com/>
28. Zuula Web meta search service, <http://www.zuula.com/>
29. Evri topic search engine, <http://www.evri.com/>
30. Hallerman, D.: Search Engine Marketing: User and Spending Trends (January 2008), <http://www.emarketer.com/Article.aspx?R=1007415>
31. Page, L.: Method for node ranking in a linked database. U.S. Patent 6,285,999, <http://www.google.com/patents?vid=6285999>
32. Brin, S., Page, L.: The anatomy of a large-scale hypertextual Web search engine. In: Proceedings of the seventh international conference on World Wide Web 7, Brisbane, Australia, pp. 107–117 (1998); (Section 2.1.1 Description of PageRank Calculation)
33. Infoseek brief history on Wikipedia,  
<http://en.wikipedia.org/wiki/Infoseek>
34. Altavista Web search service, <http://www.altavista.com/>
35. Lycos Web search service, <http://www.lycos.com/>

36. Inktomi Corporate description on Wikipedia  
[http://en.wikipedia.org/wiki/Inktomi\\_Corporation](http://en.wikipedia.org/wiki/Inktomi_Corporation)
37. Cano, P., et al.: A Review of Algorithms for Audio Fingerprinting. In: International Workshop on Multimedia Signal Processing, US Virgin Islands (December 2002)
38. Jain, A.K., Duin, R.P.W., Mao, J.: Statistical Pattern Recognition: A Review. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22(1), 4–37 (2000)
39. Bergman, M.K.: The deep web: Surfacing hidden value. *Journal of Electronic Publishing* 7(1) (August 2001)
40. Gigablast Web search service, <http://www.gigablast.com/>
41. Powerset Wikipedia semantic search service, <http://www.powerset.com/>
42. ChaCha question & answer search service, <http://www.chacha.com/>
43. Yauba real time search service, <http://www.yauba.com/>
44. Exalead demo Web search service, <http://www.exalead.com/search/>
45. Copernic Meta search service, <http://find.copernic.com/>
46. TrueKnowledge answer service, <http://www.trueknowledge.com/>

# Chapter 5:

## From Mashup Technologies to Universal Integration: Search Computing the Imperative Way

Florian Daniel, Stefano Soi, and Fabio Casati

University of Trento - Via Sommarive 14, 38123 Trento - Italy  
{daniel,soi,casati}@disi.unitn.it

**Abstract.** Mashups, i.e., web applications that are developed by integrating data, application logic, and user interfaces sourced from the Web, represent one of the innovations that characterize Web 2.0. Novel content wrapping technologies, the availability of so-called web APIs (e.g., web services), and the increasing sophistication of mashup tools allow also the less skilled programmer (or even the average web user) to compose personal applications on the Web. In many cases, such applications also feature search capabilities, achieved by explicitly integrating search services, such as Google or Yahoo!, into the overall logic of the composite application.

In this chapter, we first overview the state of the art in mashup development by looking at which technologies a mashup developer should master and which instruments exist that facilitate the overall development process. Then we specifically focus on our own mashup platform, *mashArt*, and discuss its approach to what we call universal integration, i.e., integration at the data, application, and user interface layer inside one and the same mashup environment. To better explain the novel ideas of the platform and its value in the context of search computing, we discuss an example inspired by the idea of search computing.

### 1 Introduction

The advent of Web 2.0 led to the participation of the user into the content creation and application development processes, also thanks to the wealth of social web applications (e.g., wikis, blogs, photo sharing applications, etc.) that allow users to become an active contributor of content rather than just a passive consumer, and thanks to *web mashups* [1]. Mashup tools enable fairly sophisticated development tasks inside the web browser. They allow users to develop their own applications starting from existing content and functionality. Some applications focus on integrating RSS<sup>1</sup> or Atom<sup>2</sup> feeds, others on integrating RESTful services [20], others on simple UI widgets, etc. Mashup approaches are innovative especially in that they tackle integration at the user interface level and do not “just” focus on data and in that they aim at simplicity more than robustness or completeness of features (up to the point to enable also non-professional programmers to develop own mashups).

---

<sup>1</sup> <http://cyber.law.harvard.edu/rss/rss.html>

<sup>2</sup> <http://www.ietf.org/rfc/rfc4287.txt>



Integrating content and services from the Web also means integrating *search results* or *services*, which makes mashups a natural candidate for search computing applications, but also poses novel requirements in terms of composition features – especially as for what regards UIs.

Inspired by and building upon research in SOA and capturing the trends of Web 2.0 and mashups, this chapter introduces the concept of *universal integration*, that is, the creation of composite web applications that integrate data, application, and user interface (UI) components, effectively enabling the imperative development of advanced search computing applications. Our aim is to do what service composition has done for integrating services, but to do so at all layers, not just at the application layer, and remove some of the limitations that constrained a wider adoption of workflow/service composition technologies. Universal integration can be done (and is being done) today by joining the capabilities of multiple programming languages and techniques, but it requires significant efforts and professional programmers. In this chapter we provide abstractions, models and tools so that the development and deployment of universal compositions is greatly simplified, up to the extent that even non-professional programmers can do it in their web browser.

**Scenario.** As a reference scenario throughout this chapter, we reuse the conference search scenario described in [18], based on the search query “*find all database conferences in the next six months in locations where the average temperature is 28°C degrees and for which a cheap travel solution including a luxury accommodation exists*”. Answering this request requires (i) finding interesting conferences; (ii) understanding whether the conference location is served by low-cost flights; (iii) finding luxury hotels close to the conference location with available rooms; and (iv) checking the expected average temperature of the location. Instead of automatically deriving a query plan to answer the request, in this chapter we focus on how the request can be answered through a composite application for the Web that interactively involves the user into the search process.

The screenshot in Figure 1 shows how such a *Conference Trip Planner* (CTP) application could look like. The application is composed of a variety of different components: In the upper left corner we have a *Conferences Search* component that allows the user of the application to specify a query string and to search for conferences that satisfy the query; retrieved results are displayed below the search form. This is a so-called UI component, as – besides supporting the conference search function – it also comes with its own UI, which is reused as-is by the composite application. Similarly, in the lower left corner, we have a *BBC Weather* UI component that shows the average weather conditions for a selected city, and in the upper right corner we have an *Expedia Hotel* UI component that provides a list of hotels given the name of a city. Finally, in the lower right corner, we have an *RSS Reader* UI component that displays a list of possible flight connections from Milano to the destination city.

The four UI components are synchronized via the *Conferences Search* component, which represents the entry point for the evaluation of the overall “search query”, i.e., the content displayed by the UI components. Specifically, by selecting an event of interest from the retrieved conferences, the user synchronizes the content of the other UI components in the page, resulting in a re-computation of the weather, hotel and flight components. By clicking on the proposed hotels or flights, the user is directly forwarded to the respective booking pages, where he/she can conclude the booking.

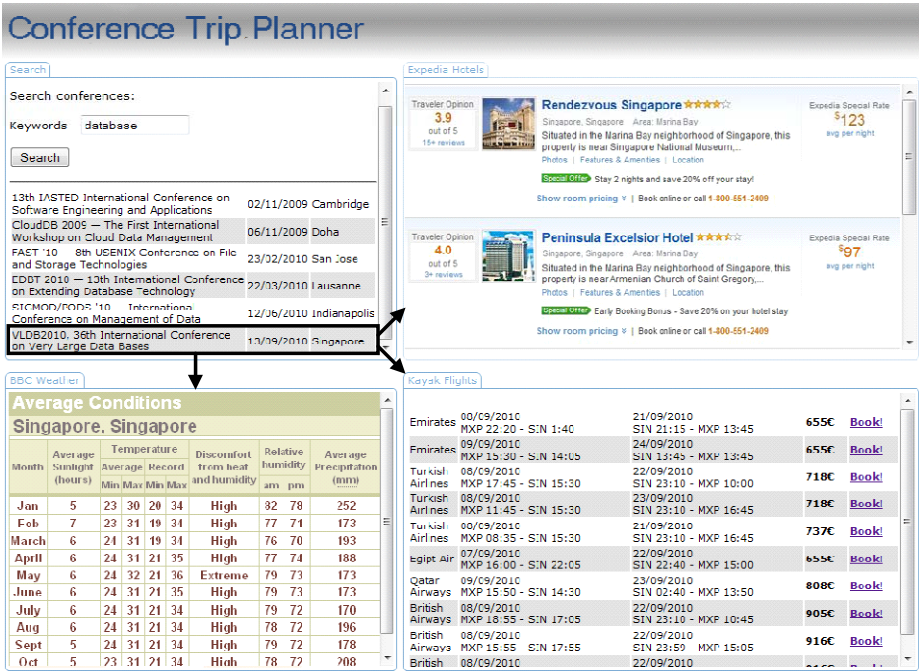


Fig. 1. Reference scenario: the conference trip planner application. Selecting a conference from the list alters the content shown by the components in the page.

We assume that the *Conferences Search* component is implemented via a simple, generic search component in conjunction with an external conference search service; in our example, we use a *Yahoo! Pipe*<sup>3</sup> to search for conferences and filter them according to the user’s query. Similarly, we use a standard *RSS Reader* component to visualize flights that are retrieved via the *kayak.com* search engine. For the *BBC Weather* and the *Expedia Hotel* components, instead, we assume that they are both provided as readily usable UI components by the respective companies.

The application in Figure 1 represents only one possible application able to answer the initial query. In fact, other combinations of components and services could be adopted, e.g., using *lufthansa.com* instead of *kayak.com* or switching the position of the weather and the hotel components, but in this chapter we are not interested in identifying the best combination of components (i.e., the best “query plan” using the terminology of [18]). The challenge we address is how to *enable the average web user to compose* an application like the one in Figure 1, relying on his/her own judgment of how components are best glued together.

**Approach and Structure of the Chapter.** In this chapter we focus on mashups and universal integration for the Web. We first offer an overview of the state of the art in traditional composition technologies (Section 2) and then specifically focus on the

<sup>3</sup> <http://pipes.yahoo.com/pipes>

recent trend of composition on the Web, i.e., mashups (Section 3). Next, we introduce the idea and principles of universal integration (Section 4). As an advanced case study and concrete implementation of the universal integration idea, in Section 5 we focus on mashArt. Specifically, we describe the conceptual and architectural aspects of mashArt, which constitute its innovative contributions in terms of *component* and *composition models* as well as *development* and *runtime* infrastructure, and show mashArt at work. Section 6 concludes the chapter.

## 2 Traditional Composition and Development Approaches

Several areas of research are related to (lightweight) composition and mashups on the Web. In this section, we briefly survey the areas of *service composition*, *UI composition*, *computer-aided web engineering tools*, *web portals and portlets*, all areas we feel particularly related to universal composition for the Web. In the next section we then put some more focus on *mashups*.

### 2.1 Service Composition Approaches

A representative of service orchestration approaches is BPEL [6], a standard composition language by OASIS. BPEL is based on WSDL-SOAP web services, and BPEL processes are themselves exposed as web services. Control flows are expressed by means of structured activities and may include rather complex exception and transaction support. Data is passed among services via variables (Java style). So far, BPEL is the most widely accepted service composition language. Although BPEL has produced promising results that are certainly useful, it is primarily targeted at professional programmers like business process developers. Its complexity (reference [6] counts 264 pages) makes it hardly applicable for web mashups.

Many variations of BPEL have been developed, e.g., aiming at invocation of REST services [7] and at exposing BPEL processes as REST services [8]. In [9] the authors describe Bite, a BPEL-like lightweight composition language specifically developed for RESTful environments. IBM's Sharable Code platform [10] follows a different strategy for the composition of REST or SOAP services: a domain-specific programming language from which Ruby on Rails application code is generated, also comprising user interfaces for the Web. In [11], the authors combine techniques from declarative query languages and services composition to support multi-domain queries over multiple (search) services, while in [21] the authors follow a document-centric approach to service composition and propose the use of AXML for service mashups. All these approaches focus on the application and data layer; UIs can then be programmed on top of the service integration logic. mashArt features instead universal integration as a paradigm for the simple and seamless composition of UI, data, and application components. We argue that universal integration will provide benefits that are similar to those that SOA and process centric integration provided for simplifying the development of enterprise processes.

## 2.2 UI Composition Approaches

In [12] we discussed the problem of integration at the presentation layer and concluded that there are no real UI composition approaches readily available: Desktop UI component technologies such as .NET CAB [13] or Eclipse RCP [14] are highly technology-dependent and not ready for the Web. Browser plug-ins such as Java applets, Microsoft Silverlight, or Macromedia Flash can easily be embedded into HTML pages; communications among different technologies remain however cumbersome (e.g., via custom JavaScript). Java portlets [15] or WSRP [2] represent a mature and Web-friendly solution for the development of portal applications; portlets are however typically executed in an isolated fashion and communication or synchronization with other portlets or web services remains hard. Portals do not provide support for service orchestration logic.

## 2.3 Computer-Aided Web Engineering Tools

In order to aid the development of complex web applications, the web engineering community has so far typically focused on model-driven design approaches. Among the most notable and advanced model-driven web engineering tools we find, for instance, WebRatio [16] and VisualWade [17]. The former is based on a web-specific visual modeling language (WebML), the latter on an object-oriented modeling notation (OO-H). Similar, but less advanced, modeling tools are also available for web modeling languages/methods like Hera, OOHDM, and UWE. All these tools provide expert web programmers with modeling abstractions and automated code generation capabilities, which are however far beyond the capabilities of our target audience, i.e., advanced web users and not web programmers.

## 2.4 Portals and Portlets

Still in the context of web applications, portals and portlets represent a different approach to the UI integration problem on the Web. Their approach explicitly distinguishes between UI components (the portlets) and composite applications (the portals) and it is probably the most advanced approach to UI composition as of today (We use the term “portlets” taken from the JSR-168 portlet specification [15], but our considerations also hold for ASP.NET Web Parts). Portlets are full-fledged, pluggable Web application components that generate document markup fragments (e.g., (X)HTML) and facilitate content aggregation in a portal server. Portlets are conceptually very similar to servlets. The main difference between them consists in the fact that while a servlet generates a complete web page, portlets generates just a piece of page (commonly called fragment) that is designed to be included into a portal page. Hence, while a servlet can be reached through a specific URL, a portlet can only be reached through the URL of the whole portal page. A portlet has no direct communication with the web browser, but this communications are managed by the portal and the portlet container that allow the request-response flows and the communication between portlets. A portal server typically allows users to customize composite pages (e.g., to rearrange or show/hide portlets) and provide single sign-on and role-based personalization.

Today, there are several standards for portlets, JSR-168 being the original specification. JSR-286 introduced inter-portlet communication via a portlet container that manages a publish-subscribe infrastructure that can be used by the portlets. Finally, WSRP [2] also added support for accessing remote portlets as web services over the Web. The portlet model is powerful as for what regards the presentation integration part, yet portals do not naturally support interactions with generic web services or the specification of orchestration logics.

### 3 Web Mashups

Web mashups somehow address the above shortcomings. Web mashups are web applications that are developed by combining content, presentation, and application functionality from disparate Web sources [1]. The term mashup typically implies easy and fast integration based on open APIs and data sources, yielding applications that add value to the individual components of the application and thereby often use components in ways that differ from the actual reason that led to the original production of the raw sources.

Mashups are strongly related with the Web. The Web is the natural environment for publishing content and services today, and therefore it is the natural environment where to access and reuse them. Content and services are published in a variety of different forms and by using a multiplicity of different technologies; we can categorize the means to source content and services from the Web into three basic groups:

- *Data services* like RSS (Really Simple Syndication) or Atom feeds, JSON (JavaScript Object Notation) or XML resources, or simple text files. A typical example is newspapers and magazines that publish their news headers via RSS or Atom feeds that allow users to easily jump to the respective articles. These simple technologies are used to publish data on the Web that are meant for consumption by machines, not humans. In fact, they focus on the efficient distribution of content, rather than on the effective presentation of such contents to human users. Sourcing data via one of these technologies is typically very simple: it mostly requires accessing an online resource and processing the response. Data services do not have complex interaction patterns to be followed.
- *Web services or public APIs accessible over the Web*, such as SOAP (Simple Object Access Protocol) or RESTful (REpresentational State Transfer) web services or, to a lower degree, Java classes (accessed via the IIOP protocol) or similar. These technologies are used to publish application logic on the Web. Their goal is therefore not just to provide access to contents or data, but also to computing logic (e.g., the processing of an order for a book shop). Typically, the interaction with web services or APIs is ruled by so-called interaction protocols, which state which operations can be invoked, in which order, by which partners, etc. Not following the rules stated by the protocol may impede the correct functioning of the service or API.
- *User interface elements*, such as HTML clips or JavaScript APIs with own user interface (e.g., Google Maps), but also banners or advertisements. Content may also be represented by already formatted and graphically rendered data (typically in HTML). In many cases, accessing such kind of content means extracting them from a web page, as there is no equivalent data service available that can be used to

source the same data. Typically, this occurs without the provider of the contents actually knowing that there is someone extracting data from its web pages. In other cases, e.g., Google Maps, the provider of the contents explicitly publishes its data at the user interface level only.

The very innovative aspect of web mashups is that they integrate sources also at the UI layer, not only at the data and application logic layers. Integration at the data and application logic layers has been extensively studied in the past, while integration at all three layers is still a goal that put architects and programmers in front of important conceptual and technical problems.

Mashup development is still an ad-hoc and time-consuming process, requiring advanced programming skills (e.g., wrapping web services, extracting contents from web sites, interpreting third-party JavaScript code, etc). There are a variety of mashup tools available online, but, as we will see, only few of them adequately address the problem of integration at all its layers. In this section, we give an overview of the state of the art in the mashup world, spanning from manual development to semi-assisted and fully assisted development approaches.

### 3.1 Manual Development

Developing applications that aggregate data, application logic and UIs coming from diverse sources requires deep knowledge about technologies like: (X)HTML, dynamic HTML, AJAX (Asynchronous JavaScript and XML), RSS, Atom; XML specifications like DTD, XSD, XSLT; protocols like SOAP or HTTP for SOAP and RESTful web services; programming languages like JavaScript, PHP, Ruby, Java, C#, and so on; relational or object-oriented databases, etc. In addition, it might be necessary to master the business protocols of employed services and to have knowledge about how to compose services into service orchestrations. This long and not exhaustive list of technologies highlights how mashing up even a simple application, such as the one in our reference scenario, is a hard and time-consuming task that can only be completed by skilled programmers.

The development of our Conference Trip Planner requires, for instance, the following skills: First of all, the developer needs to understand well the dynamics behind and interaction logic of the *Yahoo! Pipes* and *Kayak* services and the *BBC Weather*, *Expedia Hotels* and *RSS Reader* UI components of the application. In the specific case, *Expedia Hotels* and *BBC Weather* expose JavaScript APIs that allow the developer to use and interact with their services; *Pipes* and *Kayak*, instead, return their output as RSS feeds, which need to be appropriately parsed to extract all the necessary information. While the UI components already come with their own UIs, for the conference and flight search results an ad-hoc user interface has to be developed in HTML. Next, the developer needs to implement the necessary synchronization logic among the *Conferences Search* component and the others, such that on the selection of a conference the other components will coherently update their content. In addition to invoking some JavaScript functions of the UI components, this also implies interacting with the remote search services upon the selection of a conference from the list. Finally, the developer needs to create a suitable layout for the composite application, which is able to accommodate the developed components and to render the final mashup application.

The described situation is already an ideal one: all components provide some kind of componentization. If, instead, we imagine that the developer also needs to develop the components to be mashed up, things get even worse. For instance, it could be necessary to implement a wrapper for the *BBC Weather* component that is able to automatically request weather forecasts for the correct city, to extract the HTML code of the average weather conditions, and to expose a JavaScript interface that allows the interaction with other components in the application. Similar operation would be necessary also for the other components of the application.

### 3.2 Semi-assisted Development

To speed up and simplify the development especially of components to be mashed up, some useful web tools and frameworks have been recently introduced. Typically, they address the problem of data extraction from web sites and the provisioning of such data in form of data services or re-usable user interface elements. In the following, we analyze two representative tools, i.e., Dapper<sup>4</sup> and Openkapow<sup>5</sup>, which are very user-friendly.

**Dapper** is a free online instrument for the generation of data wrappers that extract data from well-structured web pages. Dapper is based on a point and click technique able to assist the user in the selection of the contents to be extracted and to infer suitable extraction rules (e.g., regular expressions). Specifically, data extraction leverages the structure of the HTML formatting to understand which elements to extract (e.g., the first cells of all the rows in a table). Once properly identified, extracted data fields can be named and structured and then published, for instance, as RSS or XML data services. Published services can easily be accessed via a unique URL and are processed each time the respective URL is accessed.

**Openkapow** is a similar open service platform based on the concept of extraction robot, that is, user-created wrappers. Users of Openkapow can build their own robots, expose their results via web services, and run them from openkapow.com for free. Robots are able to access web sites and support the extraction and reuse of data, functionality and even pieces of user interfaces. Robots are built through a visual development environment called RoboMaker. RoboMaker allows the user navigate inside the target web site and to define a series of simple steps, each one representing an event in the page, until the target data is reached. The extraction results can be exposed in two main ways: as a RESTful service or as an RSS feed, depending on the extracted content and on the expected use of it. After their publication on the Openkapow servers, robots are accessible through a public URL, which identifies the specific robot to run. So exposed services may also need some input values (e.g., user-id and password) that can be used to parameterize the services. Inputs can easily be passed by appending them to the service URL as name-value pairs, following the standard URL model.

To better understand how these tools can be used in the mashup context, let's refer again to the Conference Trip Planner example. Let us suppose that the *Kayak* flight search site does not have an RSS output for its search results. In this case, a data extraction service can be used to automatically extract the flight combinations from the

---

<sup>4</sup> <http://www.dapper.net/open/>

<sup>5</sup> <http://openkapow.com>

result page. With Dapper, for instance, a developer needs to load one or more example pages into the Dapper environment. The more example pages are loaded, the better the inferred rules. Then, the developer needs to identify the individual data items he/she wants to extract from the page by clicking on the respective HTML elements (e.g., airline, departure time, arrival time, price, intermediate stops, link to booking), to label them and to assemble the final output (e.g., an RSS feed). There is no need to write any own line of code, in order to publish the extraction results on the Web.

While this kind of tools undoubtedly speeds up the development of data extraction from existing web sites, the development effort regarding the composition of components into a new application remain in unchanged. Therefore, the developer still has to be familiar with the services and APIs to be integrated, to display sourced data in a suitable way, and to manage the communication and synchronization logic between the components. Even assuming that data extraction tools can be successfully used by non-programmers, the final mashup development therefore still remains the hard task that can be performed only by skilled programmers.

### 3.3 Fully-Assisted Development

The previous analyses and consideration show that mashup development is typically a knowledge-intensive work, involving a variety of technologies and components. In addition to simplifying the creation of data extraction instruments for web pages, which address the problem of developing *components* for mashups, it is important to also aid the actual *composition* of components into applications, which is as hard and time-consuming as developing components, if not properly supported. Mashup tools or mashup platforms address exactly this problem, each of them focusing on different composition aspects and following different mashup approaches. In the following, we analyze four of these tools, which we think are most representative for this kind of assisted mashup development: Yahoo! Pipes, JackBe Presto<sup>6</sup>, Microsoft Popfly<sup>7</sup>, and Intel Mash Maker<sup>8</sup>. There are also other tools like Google App Engine<sup>9</sup> or IBM's Lotus Mashups<sup>10</sup> and so on, but their discussion exceeds the scope of this chapter.

**Yahoo! Pipes** provides a simple and intuitive visual editor that allows one to design data-centric compositions. It takes data as input and provides data as output; the most important supported formats are RSS/Atom, XML, and JSON. A pipe is a data processing pipeline in which input data (coming from diverse data sources) are processed, manipulated and used as input for other processing steps, until the target transformation is completed. This pipeline-style process is implemented through an arbitrary number of intermediate operators, which manipulate data items inside the data feeds or provide features like loops, regular expressions or more advanced features like automatic location extraction or connection to external services. The set of operators are predefined and fixed; new functionality can be included in form of web services. Also, stored pipes can be reused as sources of another pipe.

---

<sup>6</sup> <http://www.jackbe.com/>

<sup>7</sup> <http://popflyteam.spaces.live.com> – MS Popfly has been discontinued since August 24, 2009.

<sup>8</sup> <http://mashmaker.intel.com/web>

<sup>9</sup> <http://code.google.com/intl/it-IT/appengine/>

<sup>10</sup> <http://www-01.ibm.com/software/lotus/products/mashups/>



Yahoo! Pipes' development environment is characterized by a simple and intuitive development paradigm that is however targeted at advanced web users or programmers. In fact, the level of abstraction of its operations (e.g., the regular expression component) and the characteristic data flow logic is only hardly understandable to non-programmers. Pipe's output is not meant for human consumption (RSS, Atom, JSON, etc.) but rather for integration in other applications. This limits both the variety of input sources that can be used and the accessibility of its output. In fact, the absence of any support for UIs prevents the direct use of Pipe's output by common web users. However, Pipes is a very popular data-mashup development tool, very likely due to its efficient and intuitive component placing and connection mechanism.

The development tool does not need any installation or plug-ins; it runs in any AJAX-enabled web browser. The development environment comes with a very efficient, integrated debugging tool that helps the developer during the design phase. Pipes are stored online and accessible via an own URL. When invoking a pipe, an execution process is started on the server side, relieving the client from the execution overhead. This characteristic could represent a problem under a scalability perspective: if a large number of simultaneous accesses to a pipe are made, performance and stability might suffer.

Considering our example application, with Yahoo Pipes it would be unfeasible to realize the application as described in the reference scenario, as there is no support for the user interface of the application. However, what we can do, for instance, is using Pipes to simplify the collection, aggregation and filtering of conferences sourced from different web sources, such as *conference-service.com* and *allconferences.com*. On top of this pipe, it is then necessary to provide a suitable user interface.

**JackBe Presto** is a robust and complete mashup platform which provides enterprise-level solutions. Presto gives the possibility to easily produce (design, test and deploy) mashups merging data coming from disparate sources. In particular it can be also connected to data sources very common in the business world (like Excel spreadsheets, Oracle data software, etc.), that most of mashup competitor's solutions cannot access. Simple mashup composition can be done, also by non-IT users, through the Presto Wires tool. More advanced composition can be obtained only by professional developers implementing them in EMMML language with the support of the Presto Mashup Studio plug-in for Eclipse. This language is the main actor of the OMA (Open Mashup Alliance) project, which aims to define an open language allowing enterprise mashup interoperability and portability.

The development environment is constituted by several independent tools. Wires is a visual editor based on a simple and intuitive data pipeline composition approach. It allows one to merge data coming from disparate internal and external sources producing a final output that can be graphically displayed as a mashlet. Mashlets can be plugged into a dash-board like user interface or a portal, or they can be embedded into a regular web page. Mashlet development is assisted by the Presto Mashlet tool, while the Mashup Studio is an Eclipse plug-in providing Java programmers with complete control on the mashup development process. Connectors allow one to hook up Presto to diverse software, such as Microsoft Excel, web portals, any Oracle technology, and similar. Presto services can be accessed through APIs, available for main programming languages (Java, JavaScript, C#, Python, etc.).

The runtime server provides secure mechanisms to virtualize (abstract the user from actual implementation details) and normalize (put the service output into standard formats: JSON or XML) any kind of service or data (SOAP, REST, RSS, DB, Excel) and expose them in a secure and governed way. Presto is not a hosted service, like Yahoo! Pipes; it needs to be installed and configured in each company individually.

Let us briefly analyze the possibility to create our Conference Trip Planner application with Presto. Just like Yahoo! Pipes, Wires gives the opportunity to easily access, merge and filter the RSS channels of the conferences search services and the Kayak flights search service. Retrieved items can be displayed by means of two mashlets. The development of the other UI components in form of mashlets has to be done manually in Mashup Studio using a standard programming language like Java. At this point the produced mashlets can be put together inside one web page. However, this solution does not provide for the synchronization of the basic components in the application (the mashlets), so that the selection of a conference updates the data shown in the other components. There is not inter-mashlet communication.

**Microsoft Popfly** gained a great consensus in the mashup community and achieved good levels of popularity and usage. Although the Popfly project has been discontinued, we analyze this mashup tool because we consider it an interesting example for UI composition with peculiarities that cannot be found in other tools.

Popfly provides a visual development environment for the realization of mashups based on the concept of components, or *block* as they are called in Popfly. A composition is created by dragging and dropping blocks of interest onto a design canvas and by graphically connecting them to create the desired application logic. A block can take the role of connector to external services or it can represent some internal functionality (implemented through a JavaScript function). Each block provides input and output ports that enable its connection to other blocks. Blocks can also be used to provide a user interface that can display the result of some processing. Placing multiple visualization blocks into a same page allows one to define the overall layout of the page. The internal layout of blocks can be customized by inserting ad-hoc HTML, CSS or JavaScript code. Popfly has a wide collection of available blocks, offering functionalities like RSS readers, service connectors, map components based on Virtual Earth, etc. New blocks and compositions can be defined (in JavaScript), saved, shared and managed in a dedicated section of the platform.

At runtime, the communication flow is event-driven, that is, the activation of a certain component depends on the raising of some event by another component. There is no support for exception and transaction handling, but Popfly provides a section dedicated to the test and preview of the composition. Ready compositions are stored on the Popfly server, but the execution is done on the client – as many of the built-in blocks are based on the Silverlight platform. The client-side execution of mashups alleviates the server from heavy loads and limits scalability and performance.

Considering the Conference Trip Planner application, Popfly is the first tool that can be used to fully implement the application. We assume that skilled programmers already developed and published all blocks needed for the composition, especially the UI components *Conferences Search*, *Expedia Hotels* and *BBC Weather*, while the RSS reader necessary to display the output of the conference and flight search services already exists. At this point, the developer of the composition can drag and drop these components onto the modeling canvas and connect the blocks, also providing

for the necessary mapping of the data parameters from outputs to inputs. In particular, the *Conferences Search* block must be connected to all the other blocks, in order to provide for the synchronization of the whole composition. Finally, the graphical appearance of the application's layout can be set up by including a custom CSS style sheet into the page. What is missing in Popfly is the possibility to define more complex, process-like service compositions, as could for example be needed to process the conference search results directly in Popfly.

**Intel Mash Maker** provides a completely different mashup approach: an environment for the integration of data from annotated source web pages based on a powerful, dedicated browser plug-in for the Firefox web browser. Rather than taking input from structured data sources such as RSS/Atom feeds or web services, Mash Maker allows users to reuse entire web pages and, if suitably annotated, to extract data from the pages. That is, the "components" that can be used in Mash Maker are standard web pages. If a page has been annotated in the past, it is possible to extract the annotated data from the page and share it with other components in the browser. If the page has not been annotated, it is possible reuse the page as is without however supporting any inter-page communication.

In order to annotate a page, Mash Maker allows developers and users to annotate the structure of web pages while browsing and to use such annotations to scrap contents from annotated pages. Advanced users may leverage the integrated Structure Editor to input XPath expressions with the help from FireBug's DOM Inspector (another plug-in for the Firefox web browser). Annotations are linked to target pages and stored on the Mash Maker server in order to share them with other users.

Composing mashups with Mash Maker occurs via a copy/paste paradigm, based on two modes of merging contents: *whole page merging*, where the content of one page is inserted as a header into another page; and *item-wise merging*, where contents from two pages are combined at row level, based on additional user annotations. The two techniques can be used to merge also more than two pages. Data exchange among components is achieved by means of a blackboard-like approach, where data of components integrated into an application are immediately available to all other components. Not only the development, but also the execution of mashups is entirely performed with the help from the browser plug-in at the client side; on the server side there are only the annotations for data extraction and the stored mashup definitions.

To build the Conference Trip Planner with MashMaker, first we need to devise the necessary components in form of annotated web pages. For instance, instead of using the RSS interface toward the conference search services or toward the flight search service, we need to navigate the respective web sites and annotate the data items that are necessary to answer our reference query. Similarly, we need to annotate the UI components of our application. Next, all these individual pieces of HTML markup and annotations must be joined following an item-wise merging strategy. It is possible to implement the needed synchronization mechanisms to coordinate the components of the application with each other by means of sophisticated merge operations. The whole development procedure is a non-trivial and time-consuming, it requires some non-intuitive skills to annotate, decompose, merge and reconstruct pages and web applications of arbitrary complexity. Without advanced programming skills it is hard to implement the synchronization of components upon selection of a conference.

## 4 Universal Composition: Guiding Principles

As highlighted above, although existing mashup approaches have produced promising results, techniques that cater for simple and universal integration of web components at all the three layers of the application stack are still missing. We think such techniques are necessary to transition Web 2.0 programming from elite types of computing environments to environments where users leverage simple abstractions to create composite web applications over potentially rich web components developed and maintained by professional programmers.

We aim at *universal integration*, and this has fundamental differences with respect to traditional composition. In particular, the fact that we aim at also integrating UI implies that:

- (i) *Synchronization*, and not (only) orchestration a-la BPEL, should be adopted as interaction paradigm;
- (ii) Components must be able to react to both *human user input* and *programmatically interaction*;
- (iii) We must be able to design the *user interface* of the composite application, not just the behavior and interaction among the components.

This shows the need for a model based on state, events and synchronization more than on method calls and orchestration. We recognize in particular that events, operations, a notion of state and configuration properties are all we need to model a *universal component*.

On the data side, we realize that *data integration* on the Web may also require different models: for example RSS feeds are naturally managed via a pipe-oriented data flow/streaming model (a-la Yahoo Pipes) rather than a variable-based approach as done in conventional service composition.

Another dimension of universality lies in the *interaction protocols*. As there might be a variety of components and component implementations, we must be able to deal with multiple communication protocols at the same time. For instance, the most used protocols on the Web are REST/HTTP, SOAP, RSS, Atom, and JSON.

These requirements are often at odds with the other key design goal we have: *simplicity*. We want to enable advanced web users to create applications (an old dream of service composition languages which is still somewhat a far reaching objective). This means that the universal composition paradigm must be fundamentally simpler than programming languages and current composition languages. As an example, we target the complexity of creating web pages with a web page editor, or the complexity of building a pipe with Yahoo Pipes (something that can be learned in a matter of hours rather than weeks).

## 5 The mashArt Platform

To achieve simplicity in mashArt, we make three design decisions: First, mashArt aims at *hiding* the complexity of the specific protocol or data model supported by each component. That is, the goal is that from the perspective of the composer all these specificities are hidden – with the exceptions of the aspects that have a bearing

on the composition (e.g., if a component is a feed, then we are aware that it operates, conceptually, by pushing content periodically or on the occurrence of certain events).

As a second decision, we keep the composition model *lightweight*: for example, there are no complex exception or transaction mechanisms, no BPEL-style structured activities or complex dead-path elimination semantics. This still allows a model that makes it simple to define fairly sophisticated applications. Complex requirements can still be implemented but this needs to be done in an “ad hoc” manner (e.g., through proper combinations of event listeners and component logic) but there are no specialized constructs for this. Such constructs may be added over time if we realize that the majority of applications need them.

The third decision is to focus on simplicity only *from the perspective of the user* of the components, that is, the designer of the composite applications. In complex applications, complexity must reside somewhere, and we believe that as much as possible it needs to be inside the components. Components usually provide core functionalities and are reused over and over (that’s one of the main goals of components). Thus, it makes sense to have professional programmers develop and maintain components. We believe this is necessary for the mashup paradigm to really take off. For example, issues such as interaction protocols (e.g., SOAP vs. REST or others) or initialization of interactions with components (e.g., message exchanges for client authentication) must be embedded in the components.

In the following, we describe in more detail the component model and the composition model enabling universal integration and the implementation of the mashArt platform with its design-time and runtime support.

## 5.1 The mashArt Component Model

The first step toward the universal composition model is the definition of a component model. *MashArt* components wrap UI, application, and data services and expose their features/functionalities according to the mashArt component model. The model described here extends our initial UI-only component model presented in [3] to cater for universal components. The model is based on four abstractions: state, events, operations, and properties:

- The *state* is represented as a set of name-value pairs. What the state exactly contains and its level of abstraction is decided by the component developer, but in general it should be such that its change represents something relevant and significant for the other components to know. For example, in our *Conference Search* component we can change the search string of the query and re-compute the list of pertaining conferences; this component-internal activity is irrelevant for the other components who are not interested in such low level of detail. Instead, clicking on (selecting) a specific conference expresses an information that may lead other components to show related information or application services to perform actions (e.g., query for flights). This is a state change we want to capture. In our case study, the state for the *Conference Search* component is the set of conferences being displayed plus the selected conference.

Modeling state for application components is something debatable as services are normally used in a stateless fashion. This is also why WSDL does not have a

notion of state. However, while implementations can be stateless, from a modeling perspective it can be useful to model the state, and we believe that its omission from WSDL and WS-\* standards was a mistake (with many partial attempts to correct it by introducing state machines that can be attached to service models). Although not discussed here, the state is a natural bridge between application services and data-oriented services (services that essentially manipulate a data object).

- Events communicate state changes and other information to the composition environment, also as name-value pairs. External notifications by SOAP services, callbacks from RESTful services, and events from UI components can be mapped to events. When events represent state changes, initiated either by the user by clicking on the component’s UI or by programmatic requests (through operations, discussed below), the event data includes the new state. Other components subscribe to these events so that they can change their state appropriately (i.e., they synchronize). For instance, when selecting a conference in the Conference Search component, an event is generated that carries details (e.g., name, city, start/end date) about the performed selection.
- Operations are the dual of events. They are the methods invoked as a result of events, and often represent state change requests. For example, the Conference-Search component has a state change operation ShowConferences that can be used to display retrieved conferences. In this case, the operation parameters include the necessary information about the state to which the component must evolve (the list of conferences). In general, operations consume arbitrary parameters, which, as for events, are expressed as name-value pairs to keep the model simple. Request-response operations also return a set of name-value pairs – the same format as the call – and allow the mapping of request-response operations of SOAP services, Get and Post requests of RESTful services, and Get requests of feeds. One-way operations allow the mapping of one-way operations of SOAP services, Put and Delete requests of RESTful services, and operations of UI components. The linkage between events and operations, as we will see, is done in the composition model. We found the combination of (application-specific) states, events, and operations to be a very convenient and easy to understand programming paradigm for modeling all situations that require synchronization among UI, application, or data components.
- Finally, configuration properties include arbitrary component setup information. For example, UI components may include layout parameters, while service components may need configuration parameters, such as the username and password for login. The semantics of these properties is entirely component-specific: no “standard” is prescribed by the component model.

In addition to the characteristics described above, components have aspects that are *internal*, meaning that they are not of concern to the composition designer, but only to the programmer who creates the component. In particular, a component might need to handle the invocation of a service, both in terms of mapping between the (possibly complex) data structure that the service supports and the flat data structure of mashArt (name-value pairs), and also in terms of invocation protocol (e.g., SOAP over HTTP). There are two options for this: The first is to develop ad hoc logic in form of a wrapper. The wrapper

takes the mashArt component invocation parameters, and with arbitrary logic and using arbitrary libraries, builds the message and invokes the service as appropriate. The second is to use the built-in mashArt bindings. In this case, the component description includes component bindings such as *component/http*, *component/SOAP*, *component/RSS*, or *component/Atom*. Given a component binding, the runtime environment is able to mediate protocols and formats by means of default mapping semantics. In summary, the mashArt model accommodates component models such as UI components, SOAP and RESTful services, RSS and Atom feeds.

In Figure 2(a) we introduce our graphical modeling notation for mashArt components that captures the previously discussed characteristics of components, i.e., state, events, operations, and UI. *Stateless* components are represented by circles, *stateful* components by rectangular boxes. Components with *UI* are explicitly labeled as such. We use arrows to model *data flows*, which in turn allow us to express events and operations: arrows going out from a component are *events*; arrows coming in to a component are *operations*. There might be multiple events and operations associated with one component. Depending on the particular type of operation or event of a stateless service, there might be only one incoming data flow (for one-way operations), an incoming and an outgoing data flow (for request-response operations), or only an outgoing data flow (for events). Operations and events are bound to their component by means of a simple dot-notation: *component.(operation|event)*.

The actual model of a specific component is specified by means of an abstract component *descriptor*, formulated in the *mashArt Description Language* (MDL) a simple, XML-based interface description language. MDL is for mashArt components what WSDL is for web services.

## 5.2 Universal Composition Model

Since we target universal composition with both stateful and stateless components, as well as UI composition, which requires synchronization, and service composition, which is more orchestrational in nature, the resulting model combines features from *event-based* composition with *flow-based* composition. As we will see, these can naturally coexist without making the model overly complex.

In essence, composition is defined by linking events (or operation replies) that one component emits with operation invocations of another component. In terms of flow control, the model offers conditions on operations and split/join constructs, defined by tagging operations as optional or mandatory. Data is transferred between components following a pipe/data flow approach, rather than the variables-based approach typical of BPEL or of programming languages. The choice of the data flow model is motivated by the fact that while variables work very well for programs and are well understood by programmers, data flows appear to be easier to understand for non-programmers as they can focus on the communication between a pair of components. This is also why frameworks such as Yahoo Pipes can be used by non-programmers.

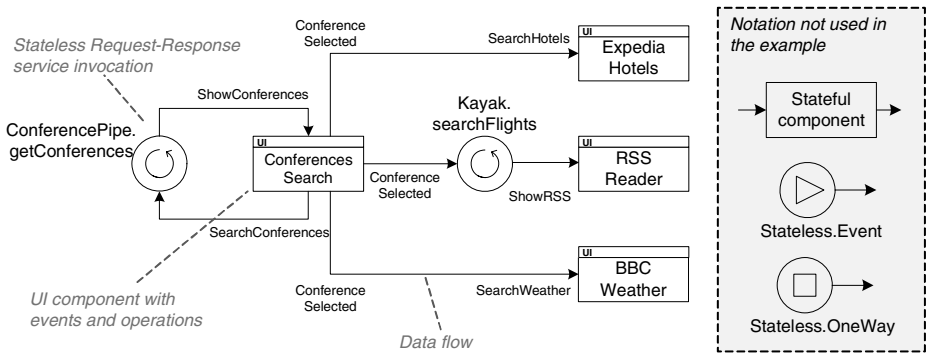
The universal composition model is defined in the Universal Composition Language (UCL), which operates on MDL descriptors only. UCL is for universal compositions what BPEL is for web service compositions (but again, simpler and for universal compositions). A universal composition is characterized by:

- *Component declarations*: Here we declare the components used in the composition and provide references to the MDL descriptor of each component and set possible constructor parameters.
- *Listeners*: Listeners are the core concept of the universal composition approach. They associate events with operations, effectively implementing simple publish-subscribe logics. Events produce parameters; operations consume them (static parameter values may be specified in the composition). Inside a listener, inputs and outputs can be arbitrarily connected (by referring to the respective IDs and parameter names) resulting into the definition of *data flows* among components. An optional condition may restrict the execution of operations; conditional statements are XPath statements expressed over the operation's input parameters. Only if the condition holds, the operation is executed.
- *Type definitions*: As for mashArt components, the structures of complex parameter values can be specified via dedicated data types.

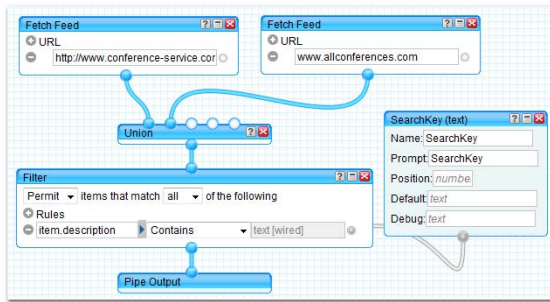
We are now ready to compose our Conference Trip Planner. Composing an application means connecting events and operations via data flows, and, if necessary, specifying conditions constraining the execution of operations. The graphical model in Figure 2(a) represents, for instance, the “implementation” of the reference scenario described in the introduction. We can see the four UI components *Conferences Search*, *Expedia Hotels*, *RSS Reader* and *BBC Weather* and the two stateless service components *ConferencePipe* and *Kayak*. The composition has four listeners:

1. If a user enters a conference search string and starts the search (*SearchConference* event), the *ConferencePipe* service is invoked by processing a Yahoo! pipe that queries two other services: *conference-service.com* and *allconferences.com*. The internals of the pipe are shown in Figure 3(b). The pipe joins the results coming from the two services and applies the filter condition provided by the user; the result is passed back to the mashArt composition by invoking the *ShowConferences* operation of the *Conferences Search* UI component.  
Note that similar operators and feed processing logics as shown in Figure 3(b) could easily be implemented also directly in mashArt, but we prefer reusing Yahoo! Pipes to show an example of how mashup platforms can interoperate.
2. If a user selects a conference from the list of retrieved conferences (*ConferenceSelected* event), three listeners reacting to the same event are activated. The first listener propagates the selected conference location and dates to the *Expedia Hotel* service that retrieves a list of available hotels from the Expedia repository.
3. The second listener activated after the selection of a conference searches for matching flights and visualizes them in the *RSS Reader*. The flights are retrieved by invoking a *kayak.com* flight search service and delivering its results as RSS feed. Such feed is provided as input to the *RSS Reader* via the *ShowRSS* operation.
4. Finally, the last listener activated upon selection of a conference aligns the data shown in the *BBC Weather* component by forwarding the name of the city the conference is located in through the *SearchWeather* operation. This causes the component to visualize the average weather conditions for the selected city.





(a) The mashArt composition model for the example scenario plus the notation not used in the example



(b) The internals of the conference search aggregation and filtering pipe

**Fig. 2.** Composition model for the Conference Trip Planner application

In the model, stateful components handle multiple invocations during their lifetime; stateless components represent single invocations. The *ConferencePipe* service is invoked each time a user inputs a new search query, while the *Conferences Search* component is instantiated only once and handles multiple events and operations.

Regarding the semantics of the three data flows leaving the *Conferences Search* component upon a *ConferenceSelected* event, it is worth noting that we allow the association of *conditions* operations. A *condition* is a Boolean expression over the operation’s input (e.g., simple expressions over name-value pairs like in *SQL where* clauses) that constrains the execution of the operation. The three data flows in Figure 2(a) represent a *parallel branch* (conjunctive semantics); if conditions were associated with either *SearchHotel*, *ShowRSS* or *SearchWeather* the flows would represent a *conditional branch* (disjunctive semantics). A similar logic applies to operations with multiple incoming flows that can be used to model *join* constructs. Inputs may be *optional* if they are not required for the execution of the operation. If only mandatory inputs are used, the semantics is conjunctive; otherwise, the semantics is disjunctive.

Data transformations can be defined via either (i) simple parameter mappings as described above; (ii) inline scripting, e.g., for the computation of aggregated or combined values; (iii) runtime XSLT transformations; or (iv) dedicated data transformation services that take a data flow in input and transform it, producing a new output.

### 5.3 Implementing and Provisioning Universal Compositions

**Development Environment.** In line with the idea of the Web as integration platform, the mashArt editor runs inside the client browser; no installation of software is required. The screenshot in Figure 3 shows how the universal composition of Figure 2(a) can be modeled in the editor. The modeling formalism of the editor slightly differs from the one introduced earlier, as in the editor we can also leverage interactive program features to enhance user experience (e.g., users can interactively choose events and operations from respective drop-down panels). But the expressive power of the editor is the same as discussed above.

The *list of available components* on the left hand side of the screenshot shows the components and services the user has access to in the online registry (e.g., the *Conferences Search* or the *BBC Weather* component). The *modeling canvas* at the right hand side hosts the composition logic represented by *UI components* (the boxes), *service components* (the circles), and *listeners* (the connectors). A click on a listener allows the user to map outputs to inputs and to specify optional input parameters.

In the lower part of the screenshot, tabs allows users to switch between different views on the same composition: visual model vs. textual UCL, interactive layout vs. textual HTML, and application preview. The layout of an application is based on standard HTML templates; we provide some default layouts, own templates can easily be uploaded. The preview panel allows the user to run the composition and test its correctness. Compositions can be stored on the mashArt server.

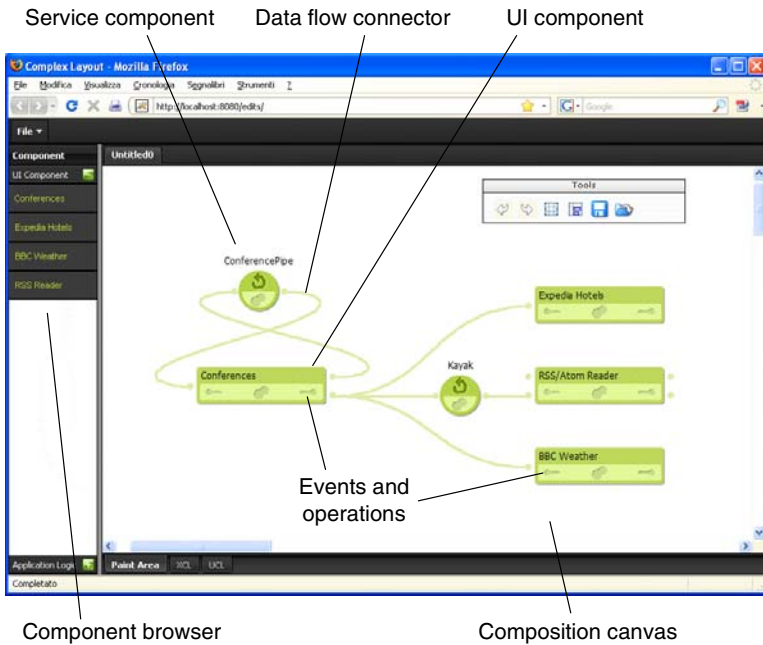
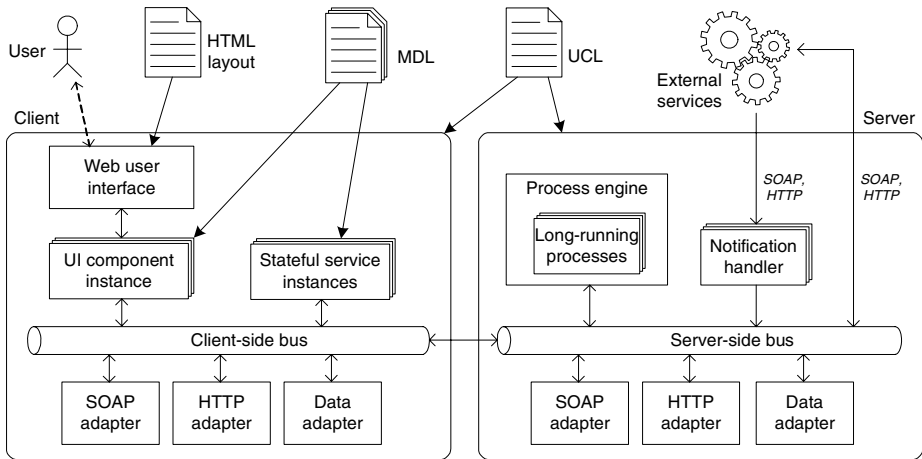


Fig. 3. The mashArt editor



**Fig. 4.** Universal execution framework

The implementation of the editor is based on JavaScript and the Open-jACOB Draw2D library (<http://draw2d.org/draw2d/>) for the graphical composition logic and AJAX for the communication between client and server. The registry on the server side, used to load components and services and to store compositions, is implemented as a RESTful web service in Java. The platform runs on Apache Tomcat.

**Execution Environment.** Developing the mashArt execution environment requires solving issues like (i) the seamless integration of stateful and stateless components and of UI and service components, (ii) the conciliation of short-lived and long-lasting business process logics in one homogeneous environment, (iii) the consistent distribution of actual execution tasks over client and server, and (iv) the transparent handling of multiple communication protocols [19].

Figure 4 illustrates the functional architecture of our execution environment. The environment is divided into a client- and a server-side part, which exchange events via a synchronization channel. On the client side, the user interacts with the application via its UI, i.e., its UI components, and thereby generates events that are intercepted by the client-side event bus. The bus implements the listeners that are executed on the client side and manage the data and SOAP-HTTP adapters. The data adapter performs data transformations, the SOAP-HTTP adapters allow the environment to communicate with external services. Stateful service instances might also use the SOAP-HTTP adapters for communication purposes.

The server-side part is structured similarly, with the difference that the handling of external notifications is done via dedicated notification handlers, and long-lasting process logics that can be isolated from the client-side listeners and executed independently can be delegated to a conventional process engine (e.g., a BPEL engine).

The whole framework, i.e., UI components, listeners, data adapters, SOAP-HTTP adapters, and notification handlers are instantiated when parsing the UCL composition at application startup. The internal configuration of how to handle the individual components is achieved by parsing each component's MDL descriptor

(e.g., to understand whether a component is a UI or a service component). The composite layout of the application is instantiated from the HTML template filled with the rendering of the application's UI components.

## 6 Conclusion

In this chapter, we have considered a novel approach to UI and service composition on the Web, i.e., *universal composition*. This composition approach is the foundation of the mashArt project, which aims at enabling even non-professional programmers (or Web users) to perform complex UI, application, and data integration tasks online and in a hosted fashion (integration as a service). Accessibility and ease of use of the composition instruments is facilitated by the simple composition logic and implemented by the intuitive graphical editor and the hosted execution environment. The platform comes with an online registry for components and compositions and will provide tools for monitoring and analysis of hosted compositions.

Throughout the chapter, we have constantly kept an eye on the connection between universal composition and *search computing*. The Conference Trip Planner tool implemented using the mashArt instruments and languages shows that it is indeed possible to develop a component-based application that provides answers to the conference search problem, provided that the necessary basic components are readily available. The application's integration logic is achieved by means of an imperative drag-and-drop composition paradigm that allows the users of the mashArt platform to compose applications according to their own knowledge about which components are needed and about how to glue them together. There exist many alternative solutions to the implementation of the same application; yet, unlike in [18], where an optimal query plan is identified automatically, in mashArt it is up to the developer to decide which solution fits best his/her individual needs.

In terms of output of the composition, it is interesting to note that while in the traditional search scenario the output is a set of result tuples, the output in mashArt is rather represented by the whole application, i.e., the individual components and their interconnection. Given the search query introduced in the introduction of this chapter, its answer is therefore represented by the screenshot in Figure 1, which naturally combines simple search outputs with sophisticated UI components.

## References

- [1] Yu, J., Benatallah, B., Casati, F., Daniel, F.: Understanding Mashup Development and its Differences with Traditional Integration. *Internet Computing* 12(5), 44–52 (2008)
- [2] OASIS. Web Services for Remote Portlets (August 2003), <http://www.oasis-open.org/committees/wsrp>
- [3] Yu, J., Benatallah, B., Saint-Paul, R., Casati, F., Daniel, F., Matera, M.: A Framework for Rapid Integration of Presentation Components. In: *WWW 2007*, pp. 923–932 (2007)
- [4] Alonso, G., Casati, F., Kuno, H., Machiraju, V.: *Web Services: Concepts, Architectures and Applications*. Springer, Heidelberg (2003)
- [5] Dustdar, S., Schreiner, W.: A survey on web services composition. *Int. J. Web Grid Services* 1(1), 1–30 (2005)

- [6] OASIS. Web Services Business Process Execution Language Version 2.0 (April 2007), <http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html>
- [7] Pautasso, C.: BPEL for REST. In: Dumas, M., Reichert, M., Shan, M.-C. (eds.) BPM 2008. LNCS, vol. 5240, pp. 278–293. Springer, Heidelberg (2008)
- [8] van Lessen, T., Leymann, F., Mietzner, R., Nitzsche, J., Schleicher, D.: A Management Framework for WS-BPEL. In: ECoWS 2008, Dublin (2008)
- [9] Curbera, F., Duftler, M.J., Khalaf, R., Lovell, D.: Bite: Workflow Composition for the Web. In: Krämer, B.J., Lin, K.-J., Narasimhan, P. (eds.) ICSOC 2007. LNCS, vol. 4749, pp. 94–106. Springer, Heidelberg (2007)
- [10] Maximilien, E.M., Ranabahu, A., Gomadam, K.: An Online Platform for Web APIs and Service Mashups. *Internet Computing* 12(5), 32–43 (2008)
- [11] Braga, D., Ceri, S., Daniel, F., Martinenghi, D.: Optimization of Multi-Domain Queries on the Web. In: VLDB 2008, Auckland, pp. 562–573 (2008)
- [12] Daniel, F., Yu, J., Benatallah, B., Casati, F., Matera, M., Saint-Paul, R.: Understanding UI Integration - A Survey of Problems, Technologies, and Opportunities. *IEEE Internet Computing*, 59–66 (May 2007)
- [13] Microsoft Corporation. Smart Client - Composite UI Application Block (December 2005), <http://msdn.microsoft.com/en-us/library/aa480450.aspx>
- [14] The Eclipse Foundation. Rich Client Platform (October 2008), <http://wiki.eclipse.org/index.php/RCP>
- [15] Sun Microsystems. JSR-000168 Portlet Specification (October 2003), <http://jcp.org/aboutJava/communityprocess/final/jsr168/>
- [16] Acerbis, R., Bongio, A., Brambilla, M., Butti, S., Ceri, S., Fraternali, P.: Web Applications Design and Development with WebML and WebRatio 5.0. *TOOLS* (46), 392–411 (2008)
- [17] Gómez, J., Bia, A., Parraga, A.: Tool Support for Model-Driven Development of Web Applications. In: Ngu, A.H.H., Kitsuregawa, M., Neuhold, E.J., Chung, J.-Y., Sheng, Q.Z. (eds.) WISE 2005. LNCS, vol. 3806, pp. 721–730. Springer, Heidelberg (2005)
- [18] Braga, D., Ceri, S., Daniel, F., Martinenghi, D.: Optimization of Multi-Domain Queries on the Web. In: VLDB 2008, Auckland, New Zealand, August 2008, pp. 562–573 (2008)
- [19] Daniel, F., Casati, F., Benatallah, B., Shan, M.-C.: Hosted Universal Composition: Models, Languages and Infrastructure in mashArt. In: Laender, A.H.F., et al. (eds.) ER 2009. LNCS, vol. 5829, pp. 428–443. Springer, Heidelberg (2009)
- [20] Fielding, R.T.: Architectural Styles and the Design of Network-based Software Architectures. University of California, Irvine, Dissertation (2000), <http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>
- [21] Abiteboul, S., Manolescu, I., Zoupanos, S.: OptimAX: efficient support for data-intensive mash-ups. In: ICDE 2008, pp. 1564–1567 (2008)

# Chapter 6:

## Web Data Extraction for Service Creation

Robert Baumgartner<sup>1</sup>, Alessandro Campi<sup>2</sup>,  
Georg Gottlob<sup>3</sup>, and Marcus Herzog<sup>1</sup>

<sup>1</sup> Lixto Software GmbH, Favoritenstrasse 9-11, 1040 Wien, Austria  
{[robert.baumgartner](mailto:robert.baumgartner@lixto.com),[marcus.herzog](mailto:marcus.herzog@lixto.com)}@lixto.com

<sup>2</sup> Politecnico di Milano, DEI, Piazza Leonardo da Vinci 32, 20133 Milano, Italy  
[campi@elet.polimi.it](mailto:campi@elet.polimi.it)

<sup>3</sup> Computing Laboratory, Oxford University, U.K.  
[gottlob@comlab.ox.ac.uk](mailto:gottlob@comlab.ox.ac.uk)

**Abstract.** Web data extraction is an enabling technique in the search computing scenario. In this chapter, we first review the state of the art in wrapper technologies focusing on how wrapper generators can be used to create unified services that integrate data from Web Applications and Web services in various domains. Next, we describe the Lixto approach and we present the Lixto Suite as one example of Web Process Integration. Finally, application areas and future challenges and the usage of wrapper technologies in the search computing context is discussed.

## 1 Introduction

Although in today's Web much data is available via APIs, light-weight and heavy-weight Web service techniques, the larger amount of data is still only available in semi-structured formats such as HTML. In the recent years, Web pages became more complex and turned into Web Applications, using a lot of Web 2.0 and Rich Internet Application technologies. As a consequence, new research and technical challenges emerged, related to automated Web navigation and data extraction.

To use Web data in Enterprise Applications and service-oriented architectures, it is crucial to provide means for automatically turning Web Applications and Web sites into Web Services, allowing structured and unified access to heterogeneous sources. This includes to understand the logic of the Web application, to fill out form values, and to grab relevant data – all these aspects need to be reflected accordingly in the generated Web Service.

In a number of business areas, Web applications are predominant among business partners for communication and business processes. Various types of processes are carried out on Web portals, covering activities such as purchase, sales, or quality management, by manually interacting with Web sites.

Wrapper Generators enable the automation of processes and operations of Web Applications. They pave the way for *Web Process Integration*, i.e. the seamless integration of Web applications into a corporate infrastructure or service oriented landscape by generating Web services from given Web sites. Web

process integration can be understood as front-end integration: integrate cooperative and non-cooperative sources without the need for information provider to change their backend. Furthermore, regarding light-weight mashup techniques, wrapper generators offer to extend the range of sources under consideration from structured Web feeds to include legacy Web applications. In this sense, Web process integration and Wrapper Technologies are essential enablers of the *Web of Services*.

The rest of the chapter is structured as follows. In Section 2, an overview of the state-of-the-art in Web data extraction methods and techniques is given. In Section 3 we give an overview of Lixto and its architecture as an example of wrapping technology used for search computing and for generating Web Process Integration scenarios. Section 4 is dedicated to survey the process of turning Web 2.0 Applications into Web Services, illustrated by describing the Lixto components and examples. Section 5 gives an overview on sample application areas as well as a summary of future research issues and the usage of Web data extraction in search computing context. Finally, some brief concluding remarks are given in Section 6.

## 2 Web Data Extraction

Web data extraction is a research field rooted in information extraction from text, in screen scrapers invented for extracting screen formatted data from main-frame applications for terminals such as VT100 or IBM 3270, and in ETL (Extract, Transform, Load) methods defined to extract information from various business processes and feed it into databases [8].

One of the first attempts to extract information from unstructured sources is [39] that presents AutoSlog, a system that automatically builds a domain-specific dictionary of concepts for extracting information from text. A significant step forward was Crystal [45], a system which allows one to automatically build a dictionary of entities from a text.

[32] presents a trial to classify wrappers considering in particular their expressiveness. Laender [34] proposed a taxonomy for data extraction tools based on the main technique used by each tool to generate a wrapper. [30] is a more recent survey on wrapper technology.

Wrapper Generation Systems can be classified according to different properties. One main such distinctive criteria is the mode of wrapper generation. This spans from manual wrapper writing (using e.g. some special-purpose APIs) to visual and interactive approaches where the user is guided through the wrapper generation and fully automated approaches. Fully automated approaches include on the one hand inductive learning based on positive and negative examples, and on the other hand unsupervised learning of similar patterns, usually restricted to a particular domain such as digital cameras. Automatic approaches tend to be limited in expressive power and robustness, but on the other hand are essential for large scale extraction scenarios.

One further differentiating criteria is the wrapper language and the objects a wrapper operates on. This ranges from perceiving wrappers as mapping

functions from node sets to node sets, various logical and automata theoretic representations, textual pattern matching on string representations of Web pages, and usage of natural language processing techniques.

The first studies dedicated to Web extraction [1,18,20,44] led the development of semi-automated systems, capable of extracting information in an automatic manner only after a training phase, performed with user intervention. TSIMMIS [23] proposes a framework for the manual construction of Web wrappers. In TSIMMIS a wrapper takes as input a specification made by a sequence of commands given by programmers describing the pages and how the data should be transformed into objects. Commands take the form (*variables, source, pattern*), where *source* specifies the input text to be considered, *pattern* specifies how to find the text of interest within the source, and *variables* are a list of variables that hold the extracted results. The generated outputs are represented using the *Object Exchange Model*. The output is composed by the target data and by additional information about the result. NoDoSE (Northwestern Document Structure Extractor) [1] is an interactive tool for semi-automatically determining the structure of such documents and then extracting their data. The user hierarchically outlines the interesting regions of files and describes their semantics. A mining component attempts to infer the *grammar* of the file from the information taken from the user. WebOQL [3] synthesizes ideas taken from query languages for the Web, from query languages for semistructured data and from languages for website restructuring. WebOQL is based on the usage of *hypertrees*, i.e., labeled ordered trees suitable to support collections, nesting, and ordering.

XWRAP [37] is a wrapper generation framework. XWRAP uses a common library to provide basic building blocks for wrapper programs. In this way, tasks of building wrappers specific to a Web source are separated from repetitive tasks for multiple sources. The wrapper building process is divided into two steps: the encoding of the source-specific metadata knowledge and the combination of the information extraction rules generated at the first phase. W4F (Wysiwyg Web Wrapper Factory) [40] is a Java toolkit to generate Web wrappers. The process is done in three steps: *retrieval, extraction, and mapping*. The first step retrieves a document from the Web and builds a DOM using an HTML parser. The next two steps apply a set of rules expressed in HEL (HTML Extraction Language) on the parse tree to extract information. Extracted information is stored using a proprietary format called NSL (Nested String List). Iepad [28] discovers extraction rules from Web pages. The system defines a data structure, called *PAT tree*, useful for the search of repeated patterns. Exploiting repeated pattern mining the system automatically identifies record boundaries.

RoadRunner [17] is based on a grammar inference techniques. It is based on an algorithm, called *match*, that exploits similarities and differences among a set of sample pages in order to infer a common grammar, which is then used as a wrapper. Previous results were obtained in Minerva [15], an attempt to exploit declarative grammar-based approaches and procedural programming in order to handle heterogeneities and exceptions. The idea is to allow the insertion of exception-handling mechanism in grammars using a special language called



*editor*. [16] defines a formal theoretical framework in which it is proved that Match runs in Ptime, whenever pages are compliant with a class of languages called Prefix Mark-up Languages. As real-life Web pages seldomly fall in this class of Languages, some studies have recently tackled the problem of improving Match in order to automatically infer a wrapper for a wider class of languages.

DEByE (Data Extraction By Example) [33] uses a small set of examples specified by the user that interacts with a tool using *nested tables* as the visual paradigm. The user defined examples are used to generate patterns which allow extracting data from new documents. For the extraction DEByE adopts a bottom-up procedure very effective with many different types of Web sources.

WARGO [38] is a system developed to allow non-technical users to generate complete wrappers for Web sources. Access to the pages containing required data is described by means of complex *Web flows* built by simply navigating with a Web browser. The *parsing* is made using interactive tool that allow users to generate complex extraction patterns by simply highlighting relevant data from very few example pages, and answering some simple questions. The system internally relies on NSEQL (Navigation SEquence Language) for specifying navigation sequences and DEXTL (Data EXtraction Language) for specifying extraction patterns.

EXALG [2] is an algorithm capable of extracting structured data from a collection of Web pages generated by encoding data from a database into a common *template*. To discover the underlying *template* that generated the pages, EXALG uses so called Large and Frequently occurring EQuivalent classes (LFEQ), i.e. sets of words that have similar occurrence pattern in the input pages. The MGS framework [24] is based on the intuition that, on the Web, the set of attributes composing an underlying schema is limited and that there is a strong overlapping between the sources. Most of the selection of the sources and part of the extraction is done by hand. The work in [31,36] describes wrapper generation with particular emphasis on their robustness. [35] proposes an approach for the automatic extraction and segmentation of records from Web tables. The proposed approach relies on a specific pattern that occurs on many Web pages for presenting lists of items: a index page containing a list of short summaries, one for each item, which include a link leading to a page about details of the specific item. Their approach leverages on the redundant information of this pattern and is based on constraint satisfaction problems and on probabilistic inference techniques.

[14] describes a system capable to populate a probabilistic database with data extracted from the Web. Data extraction is performed by TextRunner [19], an information extraction system. The massive extraction of data from the Web is the subject of *WebTables* [13,29]. However, they just concentrate on data that is published in HTML tables, and do not perform any integration of the extracted data. The work in [43] is an attempt do demonstrate that developing information extraction programs using Datalog with embedded procedural extraction predicates is a good way to proceed. Datalog provides a cleaner and more powerful

way to compose small extraction modules into larger programs. Second, query optimization can be applied to Datalog programs.

Cimple [41,42] is a system based on the interaction of an expert to provide a set of relevant sources, to design an entity relationship model describing the domain of interest, and to compose the operators for the extraction of the data from the pages. MetaQuerier [25] supports exploration and integration of databases on the Web and concentrates its contribution on exploration of the deep Web. It exploits the regularities of web forms and automatically matches interfaces.

Flint [12] automatically searches, collects and indexes Web pages publishing data representing an instance of a certain conceptual entity. Flint takes as input a small set of labeled sample pages: it automatically infers a description of the underlying conceptual entity and then searches the Web for other pages containing data representing the same entity. Flint automatically extracts data from the collected pages and stores them into a semi-structured self-describing database.

Finally, as of today, a number of commercial systems emerged, mostly in the area of interactive wrapper generation. This includes the *Denodo* ITPilot, *WebQL* (using a SQL-like query language for the Web) and *KapowTech's* Mashup Server. Commercial frameworks applying machine learning techniques include the Dapp Factory from *Dapper* and the *Fetch* Agent Plattform.

### 3 The Lixto Approach

Lixto offers state-of-the-art products for Web data extraction and integration and services for SOA-Enablement, Mashup Enablement, Market Monitoring, and Vertical Search. In this setting, we look at Lixto technology from the perspective of an enabling technology for the creation of Web process integration and search computing scenarios.

With the Lixto Visual Developer (VD), wrappers are created in an entirely visual and interactive fashion. Figure 1 sketches the architecture of VD and its runtime components.

The VD is an Eclipse-based visual integrated development environment (IDE). It embeds the *Mozilla* browser and interacts with it on various levels, e.g. for highlighting Web objects, interpreting mouse clicks, or interacting with the document object model (DOM). Usually, the application designer creates or imports a *data model* as a first step. The data model is an XML schema-based representation of the application domain.

Figure 2 gives a screenshot of the GUI of the Visual Developer. On the left-hand side, the project overview and the outline view of the currently active wrapper are illustrated. In the center, the embedded browser is shown. At the bottom, in the Property View, navigation and extraction actions can be inspected and configured (as shown in Figure 3).

During *wrapper creation*, the application designer visually creates deep Web navigations (e.g., form filling), logical elements (e.g., click if exists), and extraction rules. The system supports this process with automatic recording, immediate feedback mechanisms, and generalization heuristics. The application designer

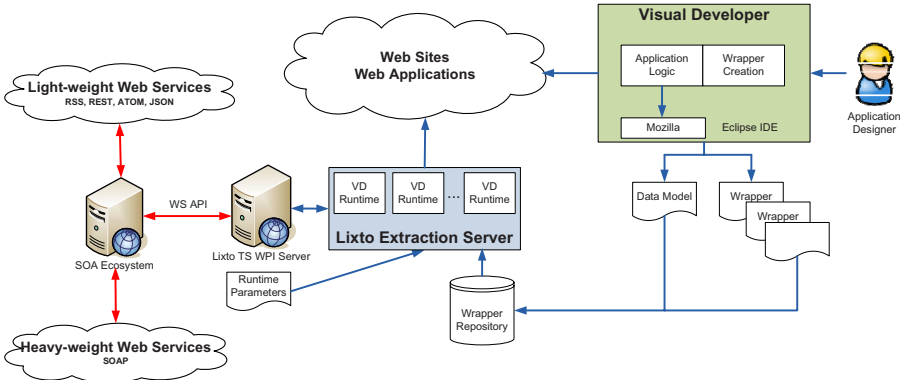


Fig. 1. Environment and Architecture



Fig. 2. Lixto Visual Developer

creates the wrapper based on samples, both in the case of navigation steps, and in the case of extraction steps. Finally, the designer parameterizes search, filtering and extraction steps of the wrapper. These parameters form the input values for the exhibited Web Service methods.

The internal extraction language *Elog* [5,22], the Web object detection based on XPath2, token grammars, and regular expressions are part of the *application logic*. Moreover, the application logic comprises deep Web navigation and workflow elements for understanding Web processes.

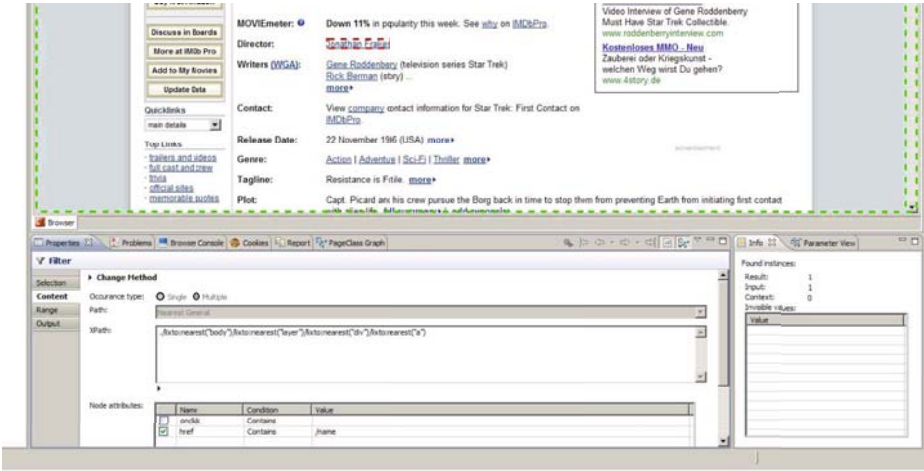


Fig. 3. Visual Filter and Condition Creation

Wrappers and data models are uploaded to the server. In the Web Process integration scenario, the WPI Edition of the Lixto Transformation Server (TS) is used (refer to Figure 1 again). In the SOA-oriented architecture of Lixto, servers such as the TS access the VD Runtimes via Web Service or Java RMI. Lixto TS exposes a query interface for ad-hoc and scheduled Web queries, and a Web Service entry-point where each request provides information about the wrapper to be executed and the runtime parameters (e.g. values for filling forms).

At wrapper execution time, each VD runtime, a.k.a. *VD head*, runs as independent process, using its own browser instance (during such executions the browser GUI is suppressed). *Lixto Extraction Server* spawns a number of VD heads and communicates results back to the server. Additionally, since Web applications can act unreliably, Extraction Server is capable of terminating and creating new heads to retry the wrapper if necessary. This architecture leverages Web extraction to a very stable and reliable process – browser instances of parallel executions do not interfere with each other, and in case of any problems with Web sites, parts of wrapper executions are retried in a new head. Due to the extraction process, Web data and Web applications can be consumed via the Lixto WPI Server as conveniently as usual light-weight and heavy-weight Web services.

## 4 Transforming Web Pages and Deep Web Sources into Web Services

In the following we exemplify the usage of Lixto components for turning a Web application into a Web service. As example we consider the IMDB site (International Movie Database <http://www.imdb.com>). The site offers information and

news about movies, tv shows, and actors. Although some information can be accessed as structured RSS feed, the majority of the data is primarily intended for manual browsing. In the following example we will extract information on particular movies, extract information about the characters and actors in the movies, and additionally extract available images about the actors. Information is extracted based on particular parameters, such as giving a movie title, specifying whether to return movies with the exact title only, return more than one movie, how many of the main characters to extract, and how many photos to include.

After defining how to extract and clean the information, which parameters can be specified and publishing it on the WPI Server, the service can be conveniently consumed by service-oriented applications.

#### 4.1 Wrapper Generation with Lixto Visual Developer

**Deep Web and Web Application Traversal.** A wrapper project in the Visual Developer comprises a number of *actions*. Actions include mouse and key events occurring during a Web navigation [4]. Such actions are e.g. link traversing, filling out textboxes, selection from lists, or opening menus. One special action is the “Data Extractor” action. Inside of this action a declarative Elog program [6] resides. In the Elog program, exit points to different pages such as “next” pages, detail pages, or dynamic changes on a page are provided – this way one can conveniently iterate over entries in selection boxes. Further actions include procedural flow controls such as if conditions, while conditions, and call actions to other page classes.

An example of a simple click action is clicking on a link to traverse to a new Web page. The corresponding action stores a generalized XPath to the corresponding link element and the information that a single mouse click is performed on this particular element. The XPath is made as robust as possible by the system to ensure a stable navigation replay even in case of changes on the Web page.

Actions are embedded in declarative templates, so-called *page classes*. In Figure 2, the wrapper outline view is shown on the left-hand side (the flow in the first page class is enlarged in Figure 4), illustrating the procedural action flow and the Elog extractors in the declarative page class templates. At the bottom, the actual page class dependencies are given.

Consider the wrapper for IMDB in Figure 2 and especially the page class dependency graph illustrated in higher resolution in Figure 5: In the “start” page class the search form is filled out and results are extracted. Moreover, based on given parameters, it is decided which elements are clicked to reach the movie detail page. For each of these, the page class “movie” is called. In this page class, the details such as director and year are extracted, as well as the most important characters and their actors. Since character and cast information is on different pages with a different structure, different data extractors are used in

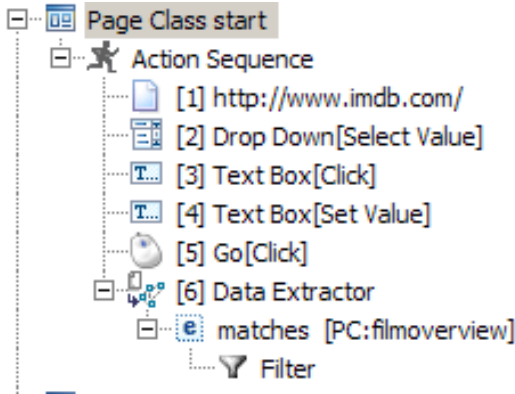


Fig. 4. A sample Page Class

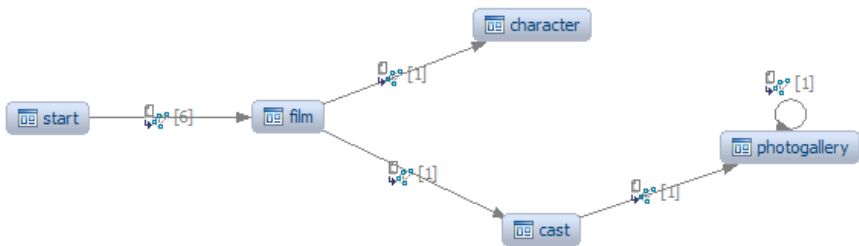


Fig. 5. IMDB Application Flow

the page class “cast” and “character”, respectively. Finally, photos of the actors can be reached from the actor page. 48 photos per page are shown, hence the page class “photo” is iteratively called until there is no next link or the given limit is reached.

Due to branching and iteration capabilities of the navigation language, complex process flows can be modeled on top of the page class concept such as e.g. a flight booking process.

**Web Data Extraction Language.** The internal data extraction language, Elog, is a datalog-like language especially designed for wrapper generation. The Elog language operates on Web objects, that are HTML elements, lists of HTML elements, and strings. Elog rules can be specified fully visually without knowledge of the Elog language. Web objects can be identified based on internal, contextual, and range conditions and are extracted as so-called “pattern instances”.

A typical Elog rule in the IMDB wrapper to extract the director of the movie looks like:

```

director(X0, X1) :-
  root(_, X0), subelem(X0,
    (./lixto:nearest("body")/lixto:nearest("layer")
     /lixto:nearest("div")/lixto:nearest("a"),
    [{"href", "name", substring}]), X1),
  before(X1, ../h5, [{"text",
    "^Director.*", regexp}]), 0, 1, X2, X3).

```

The “director” predicate used in the head of the rule evaluates to true for all assignments  $X_1$  where the body holds true. In the “subelem” predicate, for each assignment of  $X_0$  (matches of the “root” pattern) assignments for the result of the XPath generation are stored in  $X_1$ . The “before” predicate refers to instances of  $X_1$ , its results could be referenced by further predicates. The numerical values reflect distance settings (based on the node level), in this case immediately before.

Elog uses different kind of expressions to identify Web objects – this includes XPath2 statements (and extension functions) for tree nodes and regular expressions or predefined ontology concepts for textual data, and is open to be extended to e.g. extract based on the visual representation in the browser. Figure 3 illustrates how this rule is presented to wrapper designers.

Among the evaluation criteria of a wrapping language, expressiveness and robustness are the most important ones. Robustness grants that information on frequently changing Web pages are correctly discovered, even if e.g. a banner or a new page fragment is introduced. Visual Developer offers robust mechanisms of data extraction based on the two paradigms of tree and string extraction. Verification alerts can be imposed that give warnings in case user-defined criteria are no longer satisfied on a page. [22] shows a kernel fragment of *Elog* that captures monadic second order logic, hence it is very expressive while at the same time easy to use due to visual specification.

**Visual Wrapper Generation.** The usage of both Elog and of the internal Web interaction language is completely invisible to the average wrapper designer and all operations are carried out by visual means. In simple scenarios this is basically comprised of four steps:

1. First, the *modeling phase*, where the application designer defines an XML Schema-based data model to map Web data instances into or imports an existing one such as RSS.
2. As a second step, the application designer *visually records* a Web macro filling forms and traversing to the desired result page. The system protocols the actions on an action-based level, i.e. it does not rely on the server request/response, but identifies XPath elements based on user clicks in the GUI and is capable of replaying all kind of user interactions, even for highly dynamic pages.
3. Finally, the application designer designs the *data extractor* for the result page where usually hierarchically defines the elements of interests. *Filters* are



created visually by choosing example instances and then refining the selection based on system generalizations. Internally, filters are mapped to Elog rules. Result instances are mapped to the defined data model and verified for their consistency.

4. Additionally, every action and filter can be *parameterized* to individual search and restriction values, which are provided as method parameters to the Web service requests.

In real-life scenarios such as the IMDB example these steps are close by intermingled, especially when extracting data from various interlinked pages. The IMDB wrapper comprises a number of data extractors on different kind of pages, and a complex navigation describing when to apply which extractor and action. After finishing the example-based wrapper generation, certain actions and steps are manually parameterized by the designer. First, the value that is inserted into the search form, and next if one or more movie titles shall be returned based on a particular query, and how many of its actors and how many photos. In this way, similar to the output model, an input model is defined comprising all parameters that can be adjusted in an instance of an IMDB wrapping process by a request. As a next step, the wrapper is deployed to the WPI server.

## 4.2 Lixto WPI Server

**Transformation Server.** Heterogeneous environments such as integration and mediation systems require a conceptual information flow model. The usual setting for the creation of services based on Web wrappers is that information is obtained from multiple wrapped sources and has to be integrated; often source sites have to be monitored for changes, and changed information has to be automatically extracted and processed. Thus, push-based information systems architectures in which wrappers are connected to pipelines of post-processors and integration engines which process streams of data are a natural scenario, which is supported by the Lixto Transformation Server [26,7]. The overall task of information processing is composed into stages that can be used as building blocks for assembling an information processing pipeline. The stages are to

- acquire the required content from the Web applications
- integrate and transform content from a number of input channels and tasks such as finding differences,
- interact with external processes,
- format and deliver results in various formats and channels and connectivity to other systems.

The actual data flow within the Transformation Server is realized by handing over XML documents. Each stage within the Transformation Server accepts XML documents (except for the wrapper component, which accepts HTML), performs its specific task (most components support visual generation of mappings), and produces an XML document as result. This result is put to the successor components. Boundary components have the ability to activate themselves



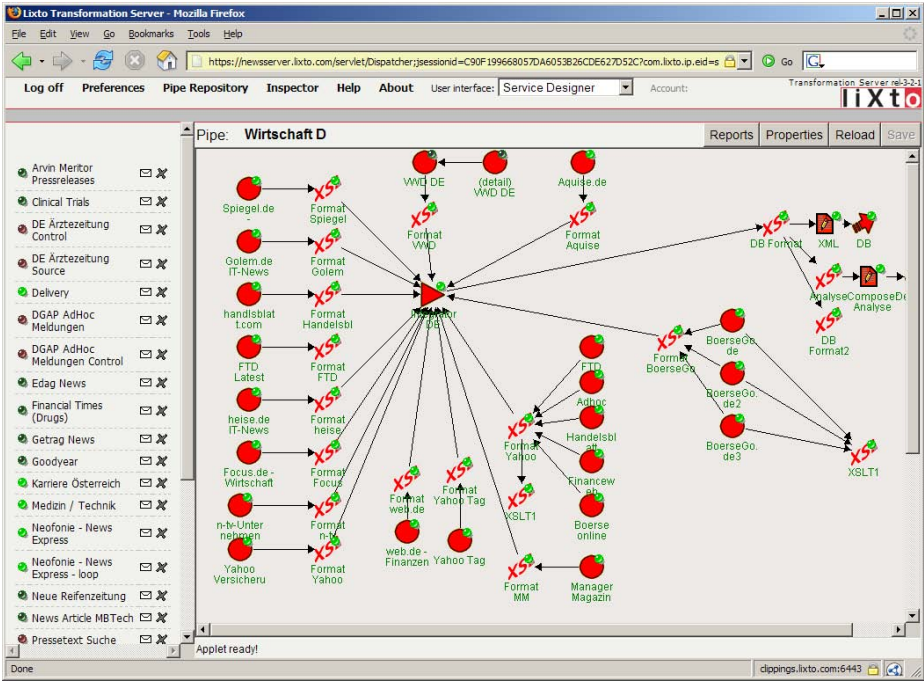


Fig. 6. Lixto Transformation Server

according to a user-specified strategy and trigger the information processing on behalf of the user. Figure 6 illustrates a complex example in the news domain.

From an architectural point of view, Lixto Transformation Server may be conceived as a container-like environment of information processing or as visually configured agent server. This “service flow” can model very complex unidirectional information flows. The usage of components also modularizes the information processing, so the service can be maintained and updated smoothly. Moreover, information services can be controlled and customized from outside of the server environment by various types of communication media such as Web services.

**Extraction Server/Cluster.** In simple scenarios, the Lixto WPI Server uses a single *Extraction Server*, where a number of extraction jobs can run in parallel. However, WPI scenarios with large number of services and users require a scalable extraction environment. It is crucial to be on the one hand very performant to support ad-hoc requests, and on the other hand to provide means for extreme scalability, especially in cases with a high peak load at certain times. Hence, data extractions can be executed via the *Extraction Cluster*. The WPI Server uses the Extraction Cluster [9] as directory service, asking for a free VD runtime head to be used in the next execution. The Extraction Cluster queues the request and assigns the best suited head, based on given weights. Furthermore, for ad

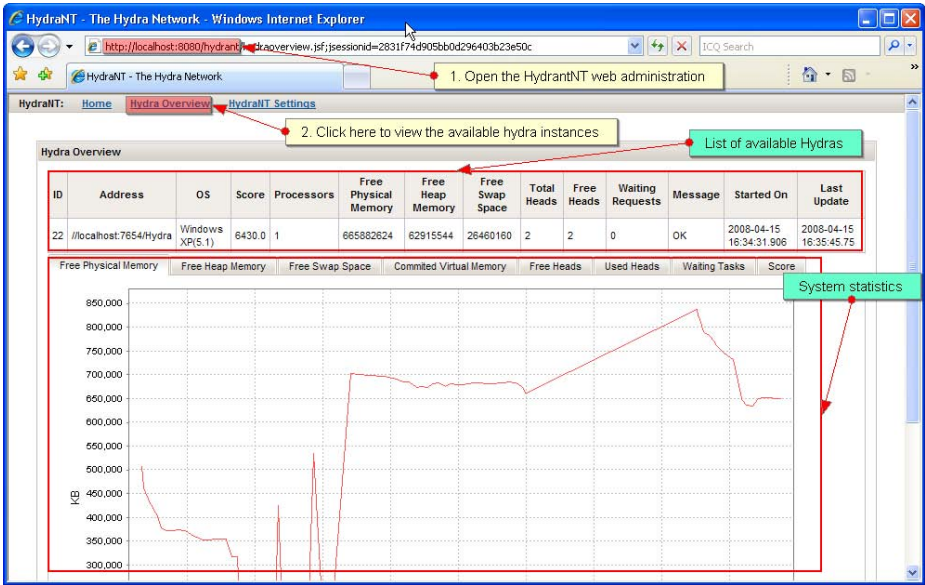


Fig. 7. Lixto Extraction Cluster

hoc requests, a priority queue is supported. Machines can be registered on the Extraction Cluster and inform it about the number of running VD heads and the machine parameters.

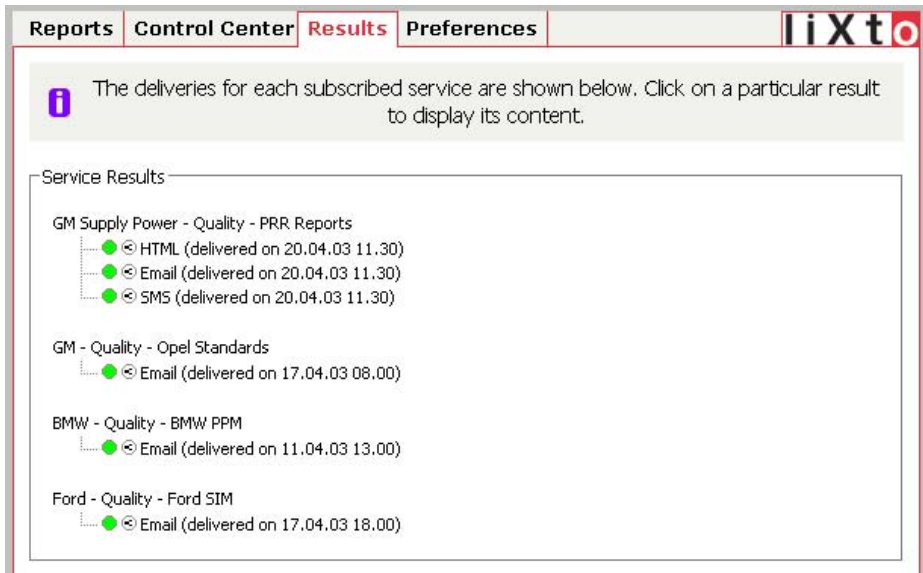
The Extraction Cluster distributes the load and can invoke Extraction Servers from Cloud Services such as the Amazon Elastic Cloud if the load gets too high. A screenshot of a simple status inspection is shown in Figure 7.

**Lixto WPI Server Users and Registry.** In Web Process Integration scenarios, we mainly distinguish two cases:

- *Scheduled Push Approach:* A service is configured to regularly push data to a particular component. The WPI Server handles the schedule and delivers results e.g. to a database or an e-mail address.
- *Ad-Hoc Pull Approach:* A service is configured to return data on demand. A Web Service interface is exposed that drives the service and executes it based on a given request. Data is returned e.g. as SOAP response or as REST.

Lixto WPI server distinguishes different user roles, the most prominent being the service designer and the service user. The service designer composes a service, including the definition of a wrapper, specification of transformation rules, how to integrate results if multiple wrappers are used, and how to deliver information. Service Designers publish services that are allowed to be consumed by Service Users.

Service users use the MyLixto GUI to browse the service registry and pick interesting services. A service user can choose subscribe to a service, which



**Fig. 8.** Consuming the WPI Service Registry with MyLixto

regularly runs in her name and with her given parameters, and provides the information e.g. through e-mail. Please refer to Figure 8 as an example. Alternatively, users can choose to receive the data immediately, triggering an execution on the WPI server. The first approach is primarily used in corporate scenarios where employees need to be informed regularly, whereas the second is usually used in meta-search scenarios and on-demand mashup applications (refer to Section 5).

### 4.3 Web Service Delivery

Figure 9 illustrates the usage of the WPI server service registry. The service registry shows all available services (company-internal and public, respectively). During service creation, the service designer chooses which query methods for a service will be available and how to map wrapper and service parameters to methods and method parameters [10].

After picking a service, the service user is shown all available methods to a service. E.g., in the IMDB case, the following methods can be exposed:

- `getSingleMovieDescription(String title)`
- `getAllMovieTitles(String searchtext)`
- `getActorDataForMovie(String title, boolean photos)`
- `getActorCharacterRelationForMovie(String title)`

As Figure 9 illustrates, there are different ways to use the service registry. Users can either use the Service User GUI (MyLixto) for triggering or subscribing to a

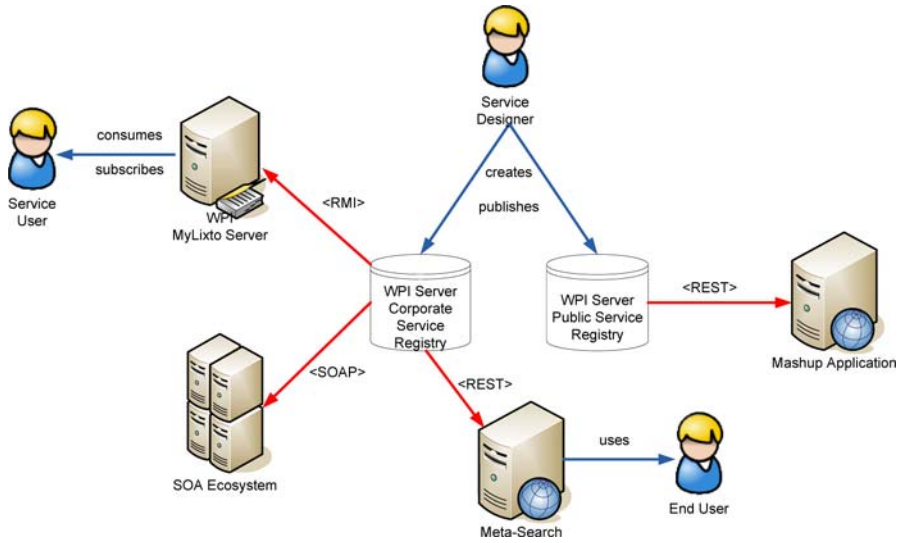


Fig. 9. Consuming the WPI Service Registry

service, or connect with their favorite Web Service client asking for the WSDL and sending a SOAP request (usually happening when the request is embedded into a larger SOA ecosystem), or if the service has been made available to the public, use a simple REST request specifying the parameters in the URL.

## 5 Application Areas and Future Research Issues

### 5.1 Sample Application Areas

**Web Process Integration in the Automotive Industry.** Many business processes in the automotive industry are carried out by means of Web portal interaction. Business critical data from various divisions such as quality management, marketing and sales, engineering, procurement, supply chain management, and competitive intelligence has to be manually gathered from Web portals and Websites. By automation, automotive part suppliers can dramatically reduce the cost associated with these processes while at the same time improving the speed and reliability with which these processes are carried out. The Automotive business case is described in more detail in [11]. In this scenario, wrapper technologies act as enabling technology for Service Oriented Architectures and are one crucial puzzle piece in Enterprise Application Integration and B2B process integration.

**End User Mashups.** Today, leading software vendors start to provide mashup platforms (such as Yahoo! Pipes or Lotus Mashups). A mashup is a Website or Web application that combines a number of Web sites into an integrated

view. Usually, the content is taken via APIs, embedding RSS or Atom Feeds in a REST-like way. Wrapper technology leverages legacy Web applications to light-weight APIs such as REST that can be integrated in mashups in the same fashion. Web Mashup Solutions no longer need to rely on APIs offered by the providers of sites, but can extend the scope to the whole Web. In particular, the deep Web gets accessible by encapsulating complex form queries and application logic steps into the methods of a Web Service. In this scenario, wrapper technologies help enable the Web of Services, built on legacy Web sites. End users are put in charge to create their own views of the Web and embed data into other applications, usually in a light-weight way. This results in “situational applications”, possibly unreliable and unsecure applications that however help to solve an urgent problem immediately.

**Vertical Flight Search and Booking.** Vertical Search is a special-purpose meta-search scenario for integrating deep Web data behind complex query interfaces and providing intelligent services to customers. Typical application scenarios are domain-specific searches with complex Web query interfaces (refer to [27] for a description how Web forms can be formally modeled), such as finding the cheapest flight over several airlines within a specific date range or the cheapest computer on various channels. Meta-Search applications have an inherent workflow logic, due to the need of querying a number of different portals and understanding dependencies when to query which Web site; e.g. querying a weather site for a particular city in a multi-hop flight scenario where first the multi-hop stops have to be extracted and understood, and next additional data for such cities is queried. Furthermore, since users do not like to wait more than a couple of seconds for results, there is the absolute need to provide results as soon as they are extracted – this logic is encapsulated in a set of Web service requests and responses. A meta-search process comprises the workflow which Web sources to query and providing input parameters to them, as well as the understanding and modeling of the Web application logic. This includes complex bi-directional processes, e.g. in cases where a booking process is re-packaged in a Meta-Search application. In such cases, interception points are required during the wrapping process.

## 5.2 Future Challenges

Turning Web Applications and Web Sites to Web Services is an important contribution to the search computing paradigm. Due to understanding of deep Web applications and parameterizing the data extraction, focused search in the Deep Web can be realized.

**Deep Web and Workflow Capabilities.** In B2B application areas, key factors are workflow capabilities for the whole process of data extraction, transformation and delivery, capabilities to treat all kinds of special cases occurring in Web interactions, and excellent support of the latest Web standards used during secure transactions.

As Web pages are becoming increasingly dynamic and interactive, efficient wrapping languages have to make it possible to record, execute and generalize macros of Web interactions and, hence, model the whole process of workflow integration. An example of such a Web interaction is a complicated booking transaction. Future research issues also include the different approach of targeted deep Web crawling as an alternative to Web application flow modelling.

To query deep Web forms, wrappers have to learn the process of filling out complex Web search forms and the usage of query interfaces. Such systems have to learn abstract representation for each search form and map them to a unified meta form and vice versa, taking into account different form element types, contents and labels.

**Extraction Capabilities.** Whereas Web wrappers today dominantly focus on either the flat HTML code or the DOM tree representation of Web pages, recent approaches aim at extracting data from the CSS box model and the visual representation of Web pages [21]. This method can be particularly useful in recent times where the DOM tree does not accurately reflect how the user perceives a Web page.

One other challenge is Generic Web Wrapping. On the one hand this includes to evolve from site-specific wrappers to domain-specific wrappers by using semantic knowledge in addition to the structural and presentational information available. On the other hand, however, it is essential that wrappers still are sufficiently robust to provide meaningful data. Hence, techniques for making wrappers more robust and automatically adapt wrappers to new situations will contribute to this challenge.

Key factors in the area of mashup scenarios include efficient real-time extraction capabilities for a large number of concurrent queries and detailed understanding of how to map queries to particular Web forms.

## 6 Conclusions

In this paper we reviewed techniques and tools for Web data extraction. We first discussed a number of tools and then focused on one particular example, the Lixto tool which is able to overcome most of these obstacles. We presented the two main components of Lixto: (1) The Lixto Visual Developer, which allows a wrapper designer to visually and interactively develop a wrapper for a Website; and (2) the Lixto Web process Integration Server (WPI Server) that enables one to quickly design an interface between complex Web processes and corporate software. In particular, we showed how Lixto can be used to transform Web pages and deep Web sources into Web services, and how massive amounts of data can be delivered into applications by means of Web process integration. The latter aspect of Web data extraction is of particular relevance to the achievements of service marts, as elaborated in Chapter 9 of this book.

We showed, based on the example of Lixto, how software of a new type can fill an important gap in information technology. While most current obstacles



are addressed and satisfactorily solved by Lixto, the Web is moving on, and new challenges emerge. Some of these challenges were described in Section 5. Other important challenges regard the intelligent and efficient querying of Web services, and the fully automatic generation of wrappers for restricted domains such as real estate, and so on. The first challenge is currently being tackled by the SeCo project. The second challenge is tackled by the DIADEM at Oxford University.

## References

1. Adelberg, B.: Nodose - a tool for semi-automatically extracting structured and semi-structured data from text documents. In: SIGMOD Record, pp. 283–294 (1998)
2. Arasu, A., Garcia-Molina, H.: Extracting structured data from web pages. In: SIGMOD 2003: Proceedings of the 2003 ACM SIGMOD international conference on Management of data, pp. 337–348. ACM, New York (2003)
3. Arocena, G.O., Mendelzon, A.O.: Webowl: restructuring documents, databases, and webs. *Theor. Pract. Object Syst.* 5(3), 127–141 (1999)
4. Baumgartner, R., Ceresna, M., Ledermüller, G.: Deep web navigation in web data extraction. In: Proc. of IAWTIC (2005)
5. Baumgartner, R., Flesca, S., Gottlob, G.: Declarative Information Extraction, Web Crawling and Recursive Wrapping with Lixto. In: Eiter, T., Faber, W., Truszczyński, M. (eds.) LPNMR 2001. LNCS (LNAI), vol. 2173, p. 21. Springer, Heidelberg (2001)
6. Baumgartner, R., Flesca, S., Gottlob, G.: Visual Web Information Extraction with Lixto. In: Proc. of VLDB (2001)
7. Baumgartner, R., Herzog, M., Gottlob, G.: Visual programming of web data aggregation applications. In: Proc. of IIWeb 2003 (2003)
8. Baumgartner, R., Gatterbauer, W., Gottlob, G.: Web data extraction system. In: Encyclopedia of Database Systems (2009)
9. Baumgartner, R., Gottlob, G., Herzog, M.: Scalable web data extraction for online market intelligence, vol. 2, pp. 1512–1523 (2009)
10. Baumgartner, R., Gottlob, G., Herzog, M., Slany, W.: Interactively Adding Web Service Interfaces to Existing Web Applications. In: Proc. of SAINT (2004)
11. Baumgartner, R., Herzog, M.: Using Lixto for automating portal-based b2b processes in the automotive industry. *International Journal of Electronic Business* 2(5), 519–530 (2004)
12. Blanco, L., Crescenzi, V., Merialdo, P., Papotti, P.: Flint: Google-basing the web. In: EDBT 2008: Proceedings of the 11th international conference on Extending database technology, pp. 720–724. ACM, New York (2008)
13. Cafarella, M.J., Halevy, A., Wang, D.Z., Wu, E., Zhang, Y.: Weatables: exploring the power of tables on the web. *Proc. VLDB Endow.* 1(1), 538–549 (2008)
14. Cafarella, M.J., Ré, C., Suciu, D., Etzioni, O., Banko, M.: Structured querying of web text: A technical challenge. In: CIDR (2007)
15. Crescenzi, V., Mecca, G.: Grammars have exceptions. *Inf. Syst.* 23(9), 539–565 (1998)
16. Crescenzi, V., Mecca, G.: Automatic information extraction from large websites. *J. ACM* 51(5), 731–779 (2004)

17. Crescenzi, V., Mecca, G., Merialdo, P.: Roadrunner: Towards automatic data extraction from large web sites. In: VLDB 2001: Proceedings of the 27th International Conference on Very Large Data Bases, pp. 109–118. Morgan Kaufmann Publishers Inc., San Francisco (2001)
18. Embley, D.W., Campbell, D.M., Jiang, Y.S., Liddle, S.W., Lonsdale, D.W., Ng, Y.k., Smith, R.D.: Conceptual-model-based data extraction from multiple-record web pages. *Data and Knowledge Engineering* 31, 227–251 (1999)
19. Etzioni, O., Banko, M., Soderland, S., Weld, D.S.: Open information extraction from the web. *Commun. ACM* 51(12), 68–74 (2008)
20. Freitag, D.: Information extraction from html: Application of a general machine learning approach. In: Proceedings of the Fifteenth National Conference on Artificial Intelligence, pp. 517–523 (1998)
21. Gatterbauer, W., Bohunsky, P., Herzog, M., Krüpl, B., Pollak, B.: Towards domain-independent information extraction from web tables. In: Proc. of WWW, May 8-12 (2007)
22. Gottlob, G., Koch, C.: Monadic Datalog and the Expressive Power of Web Information Extraction Languages. *Journal of the ACM* 51(1) (2004)
23. Hammer, J., McHugh, J., Garcia-Molina, H.: Semistructured data: The tsmimis experience. In: Proceedings of the First East-European Workshop on Advances in Databases and Information Systems, ADBIS 1997, pp. 1–8 (1997)
24. He, B., Chang, K.C.-C.: Statistical schema matching across web query interfaces. In: SIGMOD 2003: Proceedings of the 2003 ACM SIGMOD international conference on Management of data, pp. 217–228. ACM, New York (2003)
25. He, B., Zhang, Z., Chang, K.C.-C.: Towards building a metaquerier: Extracting and matching web query interfaces. In: International Conference on Data Engineering, pp. 1098–1099 (2005)
26. Herzog, M., Gottlob, G.: InfoPipes: A flexible framework for M-Commerce applications. In: Proc. of TES workshop at VLDB (2001)
27. Holzinger, W., Krüpl, B., Baumgartner, R.: Automated ontology-driven metasearch generation with metamorph. In: Vossen, G., Long, D.D.E., Yu, J.X. (eds.) WISE 2009. LNCS, vol. 5802, pp. 473–480. Springer, Heidelberg (2009)
28. Chang, C.h., Lui, S.-C.: Iepad: Information extraction based on pattern discovery, pp. 681–688 (2001)
29. Jurić, D., Banek, M., Skočir, Z.: Uncovering the deep web: Transferring relational database content and metadata to OWL ontologies. In: Lovrek, I., Howlett, R.J., Jain, L.C. (eds.) KES 2008, Part I. LNCS (LNAI), vol. 5177, pp. 456–463. Springer, Heidelberg (2008)
30. Kayed, M., Shaalan, K.F.: A survey of web information extraction systems. *IEEE Trans. on Knowl. and Data Eng.* 18(10), 1411–1428 (2006); Member-Chang, Chia-Hui and Member-Girgis, Moheb Ramzy
31. Knoblock, C.A., Lerman, K., Minton, S., Muslea, I.: Accurately and reliably extracting data from the web: a machine learning approach, pp. 275–287 (2003)
32. Kushmerick, N.: Wrapper induction: Efficiency and expressiveness. *Artificial Intelligence* 118, 2000 (2000)
33. Laender, A.H.F., Ribeiro-Neto, B., da Silva, A.S.: Debye - date extraction by example. *Data Knowl. Eng.* 40(2), 121–154 (2002)
34. Laender, A.H.F., Ribeiro-Neto, B.A., da Silva, A.S., Teixeira, J.S.: A brief survey of web data extraction tools. *SIGMOD Rec.* 31(2), 84–93 (2002)



35. Lerman, K., Getoor, L., Minton, S., Knoblock, C.: Using the structure of web sites for automatic segmentation of tables. In: SIGMOD 2004: Proceedings of the 2004 ACM SIGMOD international conference on Management of data, pp. 119–130. ACM, New York (2004)
36. Lerman, K., Minton, S.N., Knoblock, C.A.: Wrapper maintenance: a machine learning approach. *J. Artif. Int. Res.* 18(1), 149–181 (2003)
37. Liu, L., Pu, C., Han, W.: Xwrap: An xml-enabled wrapper construction system for web information sources. In: ICDE, pp. 611–621 (2000)
38. Raposo, J., Pan, A., Alvarez, M., Hidalgo, J., Vina, A.: The Wargo System: Semi-Automatic Wrapper Generation in Presence of Complex Data Access Modes. In: Proceedings of DEXA 2002, Aix-en-Provence, France (2002)
39. Riloff, E.: Automatically constructing a dictionary for information extraction tasks. In: Proceedings of the Eleventh National Conference on Artificial Intelligence, pp. 811–816. MIT Press, Cambridge (1993)
40. Sahuguet, A., Azavant, F.: Building intelligent web applications using lightweight wrappers. *Data Knowl. Eng.* 36(3), 283–316 (2001)
41. Shen, W., Derose, P., Vu, L., Doan, A., Ramakrishnan, R.: Source-aware entity matching: A compositional approach. In: IEEE 23rd International Conference on Data Engineering, ICDE 2007, pp. 196–205 (2007)
42. Shen, W., DeRose, P., McCann, R., Doan, A., Ramakrishnan, R.: Toward best-effort information extraction. In: SIGMOD 2008: Proceedings of the 2008 ACM SIGMOD international conference on Management of data, pp. 1031–1042. ACM, New York (2008)
43. Shen, W., Doan, A., Naughton, J.F., Ramakrishnan, R.: Declarative information extraction using datalog with embedded extraction predicates. In: VLDB 2007: Proceedings of the 33rd international conference on Very large data bases, pp. 1033–1044. VLDB Endowment (2007)
44. Soderland, S., Cardie, C., Mooney, R.: Learning information extraction rules for semi-structured and free text. *Machine Learning*, 233–272 (1999)
45. Soderland, S., Fisher, D., Aseltine, J., Lehnert, W.: Crystal: Inducing a conceptual dictionary. In: Mellish, C. (ed.) Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, pp. 1314–1319. Morgan Kaufmann, San Francisco (1995)

# Chapter 7:

## Dataspaces

Cornelia Hedeler<sup>1</sup>, Khalid Belhajjame<sup>1</sup>, Norman W. Paton<sup>1</sup>,  
Alessandro Campi<sup>2</sup>, Alvaro A.A. Fernandes<sup>1</sup>, and Suzanne M. Embury<sup>1</sup>

<sup>1</sup> School of Computer Science, University of Manchester, UK  
{chedeler, npaton, alvaro, embury}@cs.manchester.ac.uk

<sup>2</sup> Dipartimento di Elettronica e Informazione, Politecnico di Milano, Italy  
campi@elet.polimi.it

**Abstract.** The vision of dataspaces is to provide various of the benefits of classical data integration, but with reduced up-front costs, combined with opportunities for incremental refinement, enabling a “pay as you go” approach. As such, dataspaces join a long stream of research activities that aim to build tools that simplify integrated access to distributed data. To address dataspaces challenges, many different techniques may need to be considered: data integration from multiple sources, machine learning approaches to resolving schema heterogeneity, integration of structured and unstructured data, management of uncertainty, and query processing and optimization. Results that seek to realize the different visions exhibit considerable variety in their contexts, priorities and techniques. This chapter presents a classification of the key concepts in the area, encouraging the use of consistent terminology, and enabling a systematic comparison of proposals. This chapter also seeks to identify common and complementary ideas in the dataspaces and search computing literatures, in so doing identifying opportunities for both areas and open issues for further research.

## 1 Introduction

Data integration, in various guises, has been the focus of ongoing research in the database community for over 20 years. The objective of this activity has generally been to provide the illusion that a single database is being accessed, when in fact data may be stored in a range of different locations and managed using a diverse collection of technologies. Providing this illusion typically involves the development of a single central schema to which the schemas of individual resources are related using some form of mapping. Given a query over the central schema, the mappings, and information about the capabilities of the resources, a distributed query processor optimizes and evaluates the query.

Data integration software is impressive when it works; declarative access is provided over heterogeneous resources, in a setting where the infrastructure takes responsibility for efficient evaluation of potentially complex requests. However, in a world in which there are ever more networked data resources, data integration technologies from the database community are far from ubiquitous. This stems in significant measure from the fact that the development and maintenance of

mappings between schemas has proved to be labour intensive. Furthermore, it is often difficult to get the mappings right, due to the frequent occurrence of exceptions and special cases as well as autonomous changes in the sources that require changes in the mappings. As a result, deployments are often most successful when integrating modest numbers of stable resources in carefully managed environments. That is, classical data integration technology occupies a position at the high-cost, high-quality end of the data access spectrum, and is less effective for numerous or rapidly changing resources, or for on-the-fly data integration.

The vision of *dataspaces* [16,18] is that various of the benefits provided by planned, resource-intensive data integration should be able to be realised at much lower cost, thereby supporting integration on demand but with lower quality of integration. As a result, dataspaces can be expected to make use of techniques that infer relationships between resources, that refine these relationships in the light of user or developer feedback, and that manage the fact that the relationships are intrinsically uncertain. As such, a dataspaces can be seen as a data integration system that exhibits the following distinguishing features: (i) low/no initialisation cost, (ii) support for incremental improvement, and (iii) management of uncertainty that is inherent to the automatic integration process, but could also be present in the integrated data itself.

However, to date, no dominant proposal or reference architecture has emerged. Indeed, the dataspaces vision has given rise to a wide range of proposals either for specific dataspaces components (e.g. [23,35]), or for complete dataspaces management systems (e.g. [10,27]). These proposals often seem to have little in common, as technical contributions stem from very different underlying assumptions – for example, dataspaces proposals may target collections of data resources as diverse as personal file collections, enterprise data resources or the web. It seems unlikely that similar design decisions will be reached by dataspaces developers working in such diverse contexts. This means that understanding the relationships and potential synergies between different early results on dataspaces can be challenging; this paper provides a framework against which different proposals can be classified and compared, with a view to clarifying the key concepts in dataspaces management systems (DSMS), enabling systematic comparison of results to date, and identifying significant gaps. In the context of search computing, the chapter identifies issues that occur in dataspaces that are also relevant to search computing, such as uncertainty, and explores how dataspaces concepts may be relevant to multi-domain search and vice versa.

The remainder of the chapter is structured as follows. Section 2 describes the classification framework by introducing the various dimensions that are used to characterise data integration and dataspaces proposals. For the purpose of instantiating the framework Section 3 describes existing data integration and dataspaces proposals in the context of the classification framework. Section 4 discusses open issues for dataspaces and search computing, and Section 5 concludes the chapter. This paper extends Hedeler *et al.* [19] by extending the dimensions used in the classification, increasing the number of proposals included in the classification,

and by including a discussion of the relationship between dataspace and search computing.

## 2 The Classification Framework

Low-cost, on-demand, automatic integration of data with the ability to search and query the integrated data can be of benefit in a variety of situations, be it the short-term integration of data from several rescue organisations to help manage a crisis, the medium-term integration of databases from two companies, one of which acquired the other until a new database containing all the data is in place, or the long-term integration of personal data that an individual collects over time, e.g., emails, papers, or music. Different application contexts result in different dataspace lifetimes, ranging from short-, medium- to long-term (*Lifetime* field in Tables 1 and 2).

Figure 1 shows the conceptual life cycle of a dataspace consisting of phases that are introduced in the following. Dataspaces in different application contexts may only need a subset of the conceptual life cycle. The phases addressed are listed in *Life cycle* in Tables 1 and 2 with the initialisation, test/evaluation, deployment, maintenance, use, and improvement phases denoted as *init*, *test*, *depl*, *maint*, *use*, and *impr*, respectively.

A dataspace, just like any traditional data integration software, is initialised, which may include the identification of the data resources to be accessed and the integration of those resources. Initialisation may be followed by an evaluation and testing phase, before deployment. The deployment phase, which may not be required, for example, in the case of a personal dataspace residing on a single desktop computer, could include enabling access to the dataspace for users or moving the dataspace infrastructure onto a server. As the initialisation of a DSMS should preferably require limited manual effort, the integration may be improved over time in a pay-as-you-go manner [27] while it is being used to search and query the integrated data resources. In ever-changing environments, a DSMS also needs to respond to changes, e.g., in the underlying data resources,

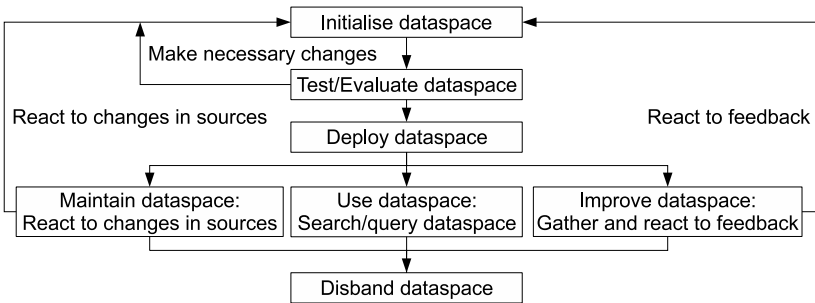


Fig. 1. Conceptual life cycle of a dataspace

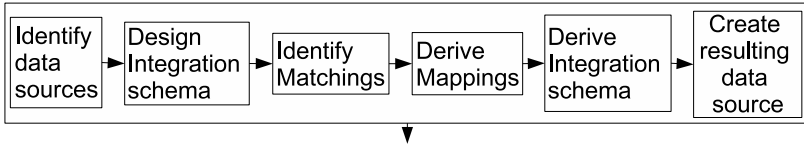


Fig. 2. Initialisation of a dataspace

which may require support for incremental integration. The phases *Use*, *Maintain* and *Improve* are depicted as coexisting, because carrying out maintenance and improvement off-line would not be desirable. For clarity, the figure does not show information flow between the different phases, so the arrows denote transitions between phases.

In the remainder of this section, the initialisation, usage, maintenance and improvement phases are discussed in more detail with a view to eliciting the dimensions over which existing dataspace proposals have varied. The dimensions are partly based on the dataspace vision [16,18] and partly on the characteristics of dataspace proposals.

## 2.1 Initialisation Phase

Figure 2 presents a more detailed overview of the steps that may be part of the initialisation phase. In the following, each of these steps is discussed in more detail and the dimensions that are used to classify the existing proposals are introduced. For each step, the dimensions are either concerned with the process (e.g., identifying matchings) and its input, or with the output of the process (e.g., the matchings identified). As others have proposed (e.g., [30]) we distinguish between *matchings*, which we take to be correspondences between elements and attributes in different schemas, and *mappings*, which we take to be executable programs (e.g., view definitions) for translating data between schemas.

**Data Sources.** A DSMS can either provide support for the integration of data sources with any kind of content (*Cont* field in Tables 1 and 2) or it can provide support for a specific application (*app-sp*), in which case assumptions that apply to that particular application can be of benefit during the initialisation phase. Utilising domain knowledge, e.g., in form of a predefined integration schema, or utilising domain knowledge during matching and mapping generation, to create domain specific dataspace solutions could result in a higher quality of the initial integration and may require less improvement. However, those solutions could be hard to port into different domains. In contrast, general purpose dataspace solutions can be applied to any domain, but may be burdened by lower quality of the integration, requiring more user feedback for improvement of the integration. General support is denoted by *gen*. Examples of specific applications include the life sciences, personal information and enterprise data. Furthermore, the data sources to be integrated can be of different types (*Type* field in Tables 1 and

2). Examples include unstructured (*unstr*), semi-structured (with no explicit schema) (*s\_str*) or structured (with explicit schema) (*str*). The data sources can also differ in their *location*: they can be local (*loc*) or distributed (*distr*).

**Integration Schema and Its Design/Derivation.** The integration schema can simply be a union schema, in which source-specific concepts are imported directly into the integration schema, or a schema that merges (e.g. [31]) the source schemas with the aim of capturing more specifically the semantic aspects that relate them. The different *types* of resulting schemas are denoted as *union* and *merge* in Tables 1 and 2, respectively. Integration schemas can also vary in their *scope*. To be able to model a wide variety of data from a variety of domains, generic models (*gen*), such as resource-object-value triples, can be used. In contrast to those, domain-specific models (*dom\_sp*) are used in other proposals. Various data models can be used to represent the integration schema. Multiple models are used by the dataspace proposals discussed here, ranging from specific models, such as the relational model, to supermodels that subsume several specific models (e.g., [2]). The *Process* of obtaining the integration schema can either be manual (*man*), i.e., it is designed, or it can be derived semi-automatically (*s\_aut*), e.g., requiring users to select between alternatives, or automatically (*aut*) without any manual intervention. A variety of information can be used as *Input* for designing or deriving the schema, which is depicted by the different locations of the *Design* and *Derive* steps in Figure 2. The schema can be designed using *schema* or instance (*inst*) information from the sources. Matchings (*match*) or mappings (*map*) can also be used as input.

**Matchings and Their Identification.** Matchings can vary with respect to their *endpoints*: they can either be correspondences between the source schemas (*src-src*) or between source schemas and the integration schema (*src-int*). The *process* of identifying the matchings can either be manual (*man*), semi-automatic (*s\_aut*) or automatic (*aut*). The identification process may require a variety of different *inputs*, e.g., the *schemas* to be matched, instances (*inst*) that conform to the schemas (which may be utilised instead of or in addition to schema information to infer matches between the schemas), and training data (*train*), e.g., when machine learning techniques are applied.

**Mappings and Their Identification.** Like matchings, mappings can also vary with respect to their *endpoints* (*src-src* or *src-int*). The *process* to derive the mappings can either be manual (*man*), semi-automatic (*s\_aut*) or automatic (*aut*). The *inputs* to the derivation process may include the *schemas* to be mapped, instances that conform to the schemas (*inst*), matchings (*match*) and/or training data (*train*), for example, when machine learning techniques are used, or a *query*.

**Resulting Data Resource.** The resulting data resources over which queries are expressed can vary with respect to their materialisation (*Materialis.*): they can either be virtual (*virt*), partially materialised (*p\_mat*) or fully materialised



**Table 2.** Properties of the initialisation, usage, maintenance, and improvement phase of existing data integration and dataspace proposals (cont.)

Dimension	Q [37]	Cimple [11,29]	CopyCat [22]	OCTOPUS [7]
<i>Life time/Life cycle</i>				
Lifetime	medium/ long	long	short	short
Life cycle	init/use/ impr	init/use/ maint/impr	init/use/ impr	init/ use/ impr
<b>Initialisation</b>				
<i>Data sources; identification</i>				
Cont	app_sp (gen)	app_sp	gen	gen
Type	s_str/str	str	s_str/str	s_str/str
Location	distr	distr	distr	distr
<i>Integration schema; design/derivation</i>				
Type	union	merge	union	merge
Scope	gen	dom_sp	dom_sp	dom_sp
Process	aut	man	s_aut	s_aut
Input	schema/ match		schema/ inst/ match	schema/ inst
<i>Matchings; identification</i>				
Endpoints	src-src	src-src/ src-int	src-src	src-int
Process	s_aut	s_aut	s_aut	s_aut
Input	schema/ inst	schema/ inst	schema/ inst	schema/ inst
<i>Mappings; identification</i>				
Endpoints	src-src	src-int	src-int	src-int
Process	aut	man	s_aut	s_aut
Input	schema/ match/ query		schema/ inst	schema/ inst
<i>Resulting data resource; creation</i>				
Materialis.	virt	p_mat	f_mat	f_mat
Reconcil.		dupl		dupl
<i>Usage: Search/query; evaluation</i>				
Specification	in_adv/ run	run	in_adv	in_adv
Type	key	key/ SPJ	VQL	key
Evaluation	compl	compl	compl	compl
Comb. res.	union	merge	union	merge
<b>Maintenance</b>				
Changes			src_sch/ src_inst	add
Reuse			int_sch	map
<b>Improvement</b>				
Approach	exp_user	exp_user	exp_user	exp_user
Stage_feedb	map	res/ res_ran	match	int_sch/ map
Stage_impr	map	map	match	int_sch/ map

(*f\_mat*). During the creation of the integrated database, duplicates (*dupl*) entities and conflicts (*confl*) can either be reconciled (*Reconciliation*) using, e.g., record linkage approaches, or be allowed to coexist.

## 2.2 Usage Phase: Search/Query and Their Evaluation

Searches and queries can be specified (*Specification*) as a workload in advance (*in\_adv*) of data integration taking place, or they can be specified after the integration, at runtime (*run*). Specifying queries in advance provides the potential for optimising the integration specifically for a particular workload. Different *types* of searches/queries can be supported by the dataspace: exploratory searches, e.g, browsing (*browse*) or visual query languages (*VQL*), which are useful either if the user is unfamiliar with the integration schema, or if there is no integration schema. Other types include keyword search (*key*), select- (*S*), project- (*P*),



join- (*J*), and aggregation (*aggr*) queries. A common aim for a dataspace is to provide some kind of search at all times [16]. Query *evaluation* can either be complete (*compl*) or partial (*part*), e.g., using top-k evaluation approaches or approaches that are able to deal with the unavailability of data sources [16]. If multiple sources are queried, the results have to be combined (*Combine results*), which may be done by forming the *union* or merging (*merge*) the results, which may include the reconciliation of duplicates entities and/or conflicts using, e.g., record linkage approaches.

### 2.3 Maintenance and Improvement Phase

The maintenance phase deals with the fact that the underlying data sources are autonomous [16], and the improvement phase aims to provide tighter integration over time [16]. The steps in both phases are comparable to the steps involved in the initialisation phase, however, additional inputs may need to be considered. Examples include user feedback, as well as previous matchings and mappings that may need to be updated after changes in the underlying schemas.

Despite the general awareness that a DSMS needs to be able to cope with evolving data sources and needs to improve over time, only limited results have been reported to date, making it hard to consolidate the efforts into coherent dimensions. In the following we suggest a set of dimensions, that may be used to characterise future research efforts (see also Tables 1 and 2).

**Maintenance:** For effective maintenance, a DSMS needs to be able to cope with a number of different *changes*, including adding (*add*) and removing (*rem*) of resources. A DSMS also needs to be able to cope with changes in the underlying sources, e.g. changes to the instances (*src\_inst*) or the schemas (*src\_sch*), as well as changes to the integration schema (*int\_sch*). Ideally, a DSMS should require little or no manual effort to respond to those changes. It may also be beneficial to *Reuse* the results of previous integration tasks, e.g., previous matchings (*match*), mappings (*map*), integration schemas (*int\_sch*), or even user feedback (*feedb*) when responding to source changes.

**Improvement:** Improvement may be achieved in a number of ways (*Approach*), including the use of different or additional approaches to those used during initialisation for deriving matchings (*a\_match*), mappings (*a\_map*), or the integration schema (*a\_int*). Furthermore, user feedback can be utilised, which could be implicit (*imp\_user*) or explicit (*exp\_user*). In cases where user feedback is considered, this could be requested about a number of different stages (*Stage\_feedb*). This includes requesting feedback on the matchings (*match*), mappings (*map*), integration schema(s) (*int\_sch*), reformulated queries (*ref\_query*), query results (*res*) (e.g., [3]) or the ranking of the results (*res\_ran*). The feedback obtained may not only be used to revise the stage about which it was acquired, but it may also be propagated for improvement at other stages (*Stage\_impr*). The values for this property are the same as for *Stage\_feedb*.

## 2.4 Uncertainty

For the purpose of this survey, we use the term uncertainty to cover various aspects, such as the trustworthiness of sources, or the robustness of algorithms which, e.g., could be represented as probabilities or scores associated with the resulting matchings or mappings. When data from a variety of sources is integrated, uncertainty may be introduced at various stages of the initialisation and maintenance phases, and may have an impact on the usage and improvement phases. As uncertainty plays a role across all phases of the dataspace life cycle, it is discussed separately here. Table 3 classifies proposals that handle uncertainty explicitly in terms of the dimensions.

When the uncertainty that is intrinsic to the integration process is made explicit, all the *concepts* that are produced during initialisation can be annotated with uncertainty information. The *concepts* include: *source data*, which in itself could be of uncertain quality; *data sources*, which for example could be ranked with respect to their relevance to a given query; the *matchings* identified, which may be computed using algorithms that use partial information; the *mappings*, which may be derived from uncertain matchings or, similar to matchings, they may be derived using incomplete information; the *integration schema*, which may be one of many alternative integration schemas that can be derived from mappings and as such may not model the conceptual world appropriately; and the *resulting data resource*, which may have uncertainty associated with its integrated content due to the uncertainty associated with the integration process itself. The uncertainty that may be accumulated throughout the various stages of the intialisation phase may then manifest itself in the usage phase. As such, *query results* or their *rankings* may be annotated with uncertainty information or quality measures. In addition, certain properties of the *query* itself may be uncertain, e.g., it may be uncertain whether a structured query that is derived from a keyword query [37] is an appropriate representation of the query the user had in mind.

Uncertainty can be represented by various *kinds of annotation*, which include: *scores*, which, for example, can be used to represent a preference; *probabilities*, which can be used to express the probability that a concept is relevant; *precision/recall* measures; or by *ranking* values without providing addition quality measures. However, the quality measures associated with a concept could have various meanings, for example, the ranking of query results could mean that the results ranked higher are more relevant to the query or that they are more likely to be part of the correct answer as the matchings, mappings, etc. that have been utilised to obtain the answers have less uncertainty associated with them than the mappings used to obtain the lower ranked results. As such, we also clarify for each proposal that represents uncertainty explicitly the *meaning* of the quality measures associated with the concepts.

The annotation representing uncertainty can be *propagated* through various of the *operations* of the initialisation phase, such as, *identify matchings*, *derive mappings*, *derive integration schema*, and *create resulting data resource*. The operations utilised during the usage phase, such as *answer query* and *combine*

**Table 3.** Handling of uncertainty by existing data integration and dataspace proposals

Concept	Proposal	Kind of annotation	Meaning	Propagation	Propag. function
Data sources	PayGo[27]	ranking	relevance to query	combine results	in-built
Matchings	OCTOPUS [7]	score/ ranking	relevance to query		
	iMeMex[10,4] iTrails[34]	prob weights	likelihood that results obtained are correct relevance of matching to query	derive mappings/ derive integration schema derive mappings/ derive integration schema	in-built in-built
Mapping	PayGo[27]	weights	distance between schemas	derive mappings/ derive integration schema	in-built
	UDI[35,13,14,36]	prob	probability that matching is correct	derive mappings/ derive integration schema/ answer query	in-built
	Roomba [23] Q [37]	score costs	confidence of match being correct bias against using matching for query as it produces worse answers from the user's point of view when used to answer a query	derive mappings/ answer query	in-built
	Cimple [11,29] CopyCat [22]	score score	confidence that match is correct relevance to integration operation	answer query derive mappings/ answer query	in-built in-built
Query	OCTOPUS [7]	score	relevance to query and table to be joined with		
Query results	Roomba [23]	score	expected result quality		
User	UDI[35,13,14,36]	score	scores of mappings used		
	Q [37]	score	cost of the query that produced result		
	Cimple [11,29] CopyCat [22]	score score	scores from matchings used score of query that produced result		
	Cimple [11,29]	score	trustworthiness of user		

*results*, can also propagate uncertainty. The *propagation function* that determines how the uncertainty is propagated can either be a predetermined *built-in* function, such as the sum or product of all the scores associated with the input concept, or can be *user-defined*, e.g., allowing the user to assign different importance in the form of weights to certain inputs. For example, users may choose to trust information coming from particular sources more than from others, and may want to encode their preference in the propagation function.

## 2.5 Human-Computer Interface

Another aspect that plays a role across the various phases of the dataspace life cycle is the Human-Computer Interface that is provided to enable the user to interact with the system (see Table 4 for properties of proposals that provide a description of their user interface, and Table 5 for properties of proposals that describe the query inputs and outputs). As users have varying backgrounds,

**Table 4.** Human-Computer Interfaces provided by existing data integration and dataspace proposals

Concept	Proposal	Kind of user	Input	Optional/ Mandatory
Matchings	Roomba [23]	domain expert	annotate	optional
Mappings	Cimple [11,29]	domain expert	provide/ edit/ annotate	mandatory/ optional
	DB2 II [17]	database expert	provide	mandatory
Integration schema	iMeMex[10,4], iTrails[34]	database expert	provide	mandatory
	Q [37]	domain expert	edit	optional
	CopyCat [22]	domain expert	provide/ edit/ annotate	mandatory
	OCTOPUS [7]	domain expert	edit/ annotate	mandatory
	SEMEX [12,25]	domain expert	edit	optional
	Cimple [11,29]	domain expert	provide	mandatory
Ranked query results	CopyCat [22]	domain expert	provide/ edit/ annotate	mandatory
	OCTOPUS [7]	domain expert	edit/ annotate	mandatory
	Q [37]	domain expert	annotate	optional

**Table 5.** Query Interfaces provided by existing data integration and dataspace proposals

Proposal	Query input	Query output
DB2 II [17]	structured	results
Aladin [24]	keywords/ structured	ranked results
SEMEX [12,25]	keywords/ structured	results/ browse
iMeMex[10,4], iTrails[34]	keywords/ structured	results/ browse/ provenance
PayGo[27]	keywords	ranked results
UDI[35,13,14,36]	structured	ranked results
Roomba [23]	keywords/ structured	results
Quarry [20]	structured	results/ browse
Q [37]	keyword	ranked results
Cimple [11,29]	keyword/ structured	ranked results/ browse
CopyCat [22]	visual query language	results/ provenance
OCTOPUS [7]	keyword	results

knowledge and experience, interfaces should be designed for different *kinds of users*. For the purpose of the classification framework, we only focus on *domain experts* who are familiar with the domain which is described by the information to be integrated and queried and *database experts* with a good understanding, for example, of the source schemas, the integration schema and how they relate to each other. Throughout the initialisation and improvement phase, users may want to or may be encouraged to provide information at various stages. The *input* provided by users may include *providing* the concepts in question as input, *editing* those suggested by the integration system, or *annotating* them with quality measures, for example, by indicating which were expected by the user (true positives), or which were not expected (false positives). *Concepts* that users may provide as input, edit or annotate include *matchings*, *mappings*, the *integration schema*, *query results* and *ranked query results*. Providing the information or annotation can either be *mandatory* or *optional*.

To cater for different kinds of users but also different degrees of integration, different interfaces for querying the integrated sources as well as viewing and possibly exploring the results may have to be provided. The *query input* can be provided using a *visual query language*, *keywords* or *structured queries*, such as

the select, project and join queries mentioned earlier in the usage phase. The *query output* could consist simply of the *results* or the *ranked results*, or could in addition include some *provenance* information that can be explored by the user to identify the source of the information returned. In addition, a means may be provided to *browse* the results and their associations with other information, for example by providing links that users can follow.

### 3 Data Integration Proposals

For the purpose of comparison, this section uses the framework to characterise and describe a number of dataspaces proposals, and in addition the data integration facilities of DB2 [17] as an example of a classical data integration approach. The proposals were classified according to the dimensions in Section 2. Only published proposals were chosen for which sufficient implementation detail is available to enable them to be classified according to the framework presented in Section 2.

DB2 [17] follows a database federation approach. It provides uniform access to heterogeneous data sources through a relational database that acts as mediation middleware. The integration schema could be a *union* schema, or a *merged* schema defined by views which need to be written *manually*. Data sources are accessed by wrappers, some of which are provided by DB2 and some of which may have to be written by the user. A wrapper supports full SQL and translates (sub)queries of relevance to a source so that they are understood by the external source. Due to the *virtual* nature of the resulting data resource, changes in the underlying data sources may be responded to with limited manual effort. In summary, DB2 relies on largely manual integration, but can provide tight semantic integration and powerful query facilities in return.

ALADIN [24] supports semi-automatic data integration in the life sciences, with the aim of easing the addition of new data sources. To achieve this, ALADIN makes use of assumptions that apply to this domain, i.e., that each database tends to be centered around one primary concept with additional annotation of that concept, and that databases tend to be heavily cross-referenced using fairly stable identifiers. ALADIN uses a *union* integration schema, and predominantly *instance-based* domain-specific approaches, e.g., utilising cross-referencing to discover relationships between attributes in entities. The resulting links are comparable to *matchings*. *Duplicates* are discovered during *materialisation* of the data resource. Links and duplicate information are utilised for *exploratory* and *keyword* searches and may help life scientists to discover previously unknown relationships. To summarise, ALADIN provides fairly loose integration and mainly exploratory search facilities that are tailored to the life sciences domain.

SEMEX [12,25] integrates personal information. A domain model, which essentially can be seen as a *merged* integration schema, is provided *manually* upfront, but may be extended manually if required. Data sources are accessed using wrappers, some provided, but some may have to be written manually. The schemas of the data sources are *matched* and *mapped automatically* to the domain

model, using a bespoke mapping algorithm that utilises heuristics and *reuses* experience from previous matching/mapping tasks. As part of the *materialisation* of the resulting data resource, *duplicate* references are reconciled, making use of domain knowledge, e.g., exploiting knowledge of the components of email addresses. SEMEX provides support for *adding* new data sources and *changes* in the underlying data, e.g., people moving jobs and changing their email address or phone number, which require domain knowledge to be resolved, e.g., to realise that it is still the same person despite the change to the contact details. SEMEX, therefore, can be seen as a domain-specific dataspace proposal that relies on domain knowledge to match schemas to the given integration schema and reconcile references automatically.

iMeMeX [10,4] is a proposal for a dataspace that manages personal information; in essence, data from different sources such as email or documents are accessed from a graph data model over which path-based queries can be evaluated. iMeMeX provides low-cost data integration by initially providing a *union* integration schema over diverse data resources, and supports incremental refinement through the *manual* provision of path-based queries known as iTrails [34]. These trail definitions may be associated with a *score* that indicates the *uncertainty* of the author that the values returned by an iTrail is correct. As such, iMeMeX can be seen as a light weight dataspace proposal, in which uniform data representation allows queries over diverse resources, but without automation to support tasks such as the management of relationships between sources.

PayGo [27] aims to model web resources. The schemas of all sources are integrated to form a *union* schema. The source schemas are then matched *automatically* using a schema matching approach that utilises results from the matching of large numbers of schemas [26]. Given the similarity of the schemas determined by matching, the schemas are then clustered. *Keyword* searches are reformulated into structured queries, which are compared to the schema clusters to identify the relevant data sources. The sources are ranked based on the similarity of their schemas, and the results obtained from the sources are *ranked* accordingly. PayGo [27] advocates the *improvement* of the semantic integration over time by utilising techniques that automatically suggest relationships or incorporate user feedback; however, no details are provided as to how this is done. In summary, PayGo can be seen as a large-scale, multi-domain dataspace proposal that offers limited integration and provides keyword-based search facilities.

UDI [35,13,14,36] is a dataspace proposal for integration of a large number of domain independent data sources automatically. In contrast to the proposals introduced so far, which either start with a manually defined integration schema or use the union of all source schemas as integration schema, UDI aims to derive a *merged* integration schema *automatically*, consolidating schema and instance references. As this is a hard task, various simplifying assumptions are made: the source schemas are limited to relational schemas with a single relation, and for the purpose of managing uncertainty, the sources are assumed to be independent. Source schemas are matched *automatically* using existing schema matching techniques [32]. Using the result of the matching and information on which

attributes co-occur in the sources, attributes in the source schemas are clustered. Depending on the *scores* from the matching algorithms, matchings are deemed to be certain or uncertain. Using this information, multiple mediated schemas are constructed, which are later consolidated into a single *merged* integration schema that is presented to the user. Mappings between the source schemas and the mediated schemas are derived from the matchings and have *uncertainty* measures associated with them. Query results are *ranked* based on the scores associated with the mappings used. In essence, UDI can be seen as a proposal for automatic bootstrapping of a dataspace, which takes the uncertainty resulting from automation into account, but makes simplifying assumptions that may limit its applicability.

Even though the majority of proposals acknowledge the necessity to improve a dataspace over time, Roomba [23] is the first proposal that places a significant emphasis on the *improvement* phase. It aims to improve the degree of semantic integration by asking users for *feedback* on *matches* and *mappings* between schemas and instances. It addresses the problem of choosing which matches should be confirmed by the user, as it is impossible for a user to confirm all uncertain matches. Matches are chosen based on their utility with respect to a *query* workload that is provided *in advance*. To demonstrate the applicability of the approach, a *generic* triple store has been used and *instance-based matching* using string similarity is applied to obtain the matches.

Quarry [20] also uses a *generic* triple store as its resulting data source, into which the data is *materialised*. Using a *union* schema, the data from the data sources coexists without any semantic integration in the form of matchings or mappings. So called signature tables, which contain the properties for each source, are introduced and it is suggested that signature tables with similar properties could be combined. Quarry provides an API for browsing the integrated data and for posing select and project queries.

Q [37], the query system of ORCHESTRA [21], a collaborative data sharing system, covering the three phases initialisation, usage and improvement, uses a *generic* graph structure to store the schemas and *matches* between schema elements, which are derived *semi-automatically* and *annotated* with costs representing the bias of the system against using the matches. *Mappings* in the form of query templates are derived from keyword queries posed by the user and matched against the schemas and matches. Multiple mappings are ranked by the sum of the cost associated with the matches utilised for the mapping, and may be edited and made persistent by the user for further reuse and parameterisation by him and other users. The parameterised queries are executed and the results ranked by the cost associated with the query that produced them and annotated with *provenance* information that enables the propagation of user feedback from the ranked query results to the corresponding mapping. Users may provide *feedback* on the results and their ranking, indicating which results should be removed, or how results should be ranked, which is propagated to the ranking of the producing mappings and the costs of the matchings utilised. The feedback is used to



adjust the costs of the matchings and thus the ranking of the mappings used to answer the query in an attempt to try and learn the model the user has in mind.

Community Information Management systems, such as Cimple [11,29] aim to reduce the up-front cost of data integration by leveraging user feedback from the community. An integration schema is provided *manually*, sources *matched* in a *semi-automatic* manner in which an automatic tool is used as a starting point and users are asked to answer questions, thus confirming or rejecting matches suggested by the automatic tool. The *uncertainty* associated with the matches is propagated through to the query results, which are *annotated* with scores and *ranked*. As Cimple applies a mass collaboration approach and aims to reduce the uncertainty by gathering feedback from users, it is aware of trustworthy and untrustworthy users providing feedback, something not taken into account by other proposals that gather user feedback. It handles feedback by the different classes of users by ignoring feedback from untrustworthy users and taking a majority vote on the feedback from trustworthy users to identify correct matches.

CopyCat [22] follows a more interactive approach to data integration, combining the integration-, usage- and improvement phases by providing a spreadsheet-like workspace in which users copy and paste examples of the data they would like to integrate to answer the queries they have. The user copies data instances from various sources into the spreadsheet, thus specifying the *integration schema* and *mappings* initially manually. The system then tries to learn the schemas of the sources and the semantic types of the data from those examples and uses the learned information to identify *matches* between sources and to suggest *mappings* that reproduce the example tuples provided by the user or that integrate further data, thus making it a *semi-automatic* integration process. Users can provide *feedback* on those suggestions by either ignoring, accepting, *editing*, or *providing* alternatives. To ease the decision process for the user, *provenance* information is provided with the suggested data to be integrated. The user feedback is propagated back through the mappings to the matchings and their scores adjusted accordingly to reflect the user preference which in turn affects the scores and, therefore, the rankings of the mappings.

Similar to CopyCat, OCTOPUS [7] provides the means for integrating multiple sources on the web interactively by providing several operations that can be utilised to create an integrated data source. Using the SEARCH operator, the user states a *keyword query*, for which the system tries to find sources which are *ranked* according to their relevance with respect to the query. If multiple data sources are required to gather the required information, users can use the EXTEND operator, providing a column of a table with which to join the new table and a keyword stating the information desired. With that information the system tries to find appropriate source tables which are *ranked* according to their relevance with respect to the query and their compatibility with the column provided as input. Throughout the whole integration process, users can provide *feedback* by *editing* or *annotating* in form or rejecting or accepting the suggested source tables.



Both, CopyCat and OCTOPUS do not distinguish between the various phases of the dataspaces lifecycle, e.g., initialisation, usage, and improvement. Instead, they promote a seamless combination of initialisation, usage and improvement of the dataspaces, albeit with a fair amount of user input required.

## 4 The Interplay between Dataspaces and Search Tasks

In this section, we investigate the interplay between dataspaces and search tasks. In particular, we show how features that are peculiar to search tasks can be borrowed and adapted in a dataspaces context and vice-versa, and pinpoint open issues that arise as a result.

### 4.1 Performing Search Tasks in Dataspaces

One of the defining features of search tasks is that the sources return streams of ranked results. We refer to sources of this kind using the term *search sources*. Although one could imagine dataspaces queries that involve search sources, the classification and survey presented earlier in this chapter show that existing dataspaces proposals do not support them. This raises the question as to how a dataspaces system can be adapted to support queries over search sources. In what follows, we discuss issues that have to do with the initialisation and improvement phases of dataspaces when user queries are answered using search sources.

*Usage.* To support search sources, the query processor of the dataspaces system needs to be able to produce results by combining streams of sub-results produced by multiple search sources. Furthermore, the results obtained need to be ranked in the light of the rankings of the sub-results produced by the search sources. In this respect, techniques from the search computing field can be borrowed and adapted for combining and ranking dataspaces query results.

*Improvement.* To perform a search task, the dataspaces system needs mappings from the integration schema, which is used by the user to pose queries, to the schemas of search sources. In doing so, the system needs to identify the search sources of relevance to users' queries. The identification of search sources can be performed in incremental manner by seeking feedback from users, e.g., the user can specify whether a result that is obtained from given search sources meets the expectations.

### 4.2 Using Dataspaces for Multi-domain Search Tasks

Multi-domain search tasks involve retrieving and combining the results obtained from multiple search sources. In what follows, we discuss issues that arise in the context of multi-domain searches, and shows how they can be addressed by adopting the pay-as-you-go philosophy adopted in dataspaces.

*Query Expression.* In a dataspace, the schemas of local data sources are initially integrated using low cost techniques, in particular, schema matching and schema merging algorithms are used for mappings the sources schemas and creating the integration schema (see the *integration schema* dimensions). The system is then improved in the light of feedback provided by the user in an incremental manner. One could envisage adopting a similar approach for easing the specification of multi-domain searches. In particular, the specification of the connections between the search services involved can be automatically derived using, e.g., matching techniques. Because those connections are derived based on heuristics, they may not meet the designer's expectations, which gives rise to the following research issue: *How can the connections suggested by inference tools to link search sources be verified?*

Another issue that arises in search tasks is the specification of queries that capture user's expectations. In dataspace, the user can pose a structured query, e.g., using SQL, or specify a collection of keywords from which the dataspace system attempts to construct/learn a structured query using as input the source schemas and the mappings that connect the elements of these schemas [28,38] (see the *query type* dimension). *Can a similar approach be adopted for specifying queries in the context of search tasks?* One could envisage the case in which the user specifies a form that captures the elements of the search results the user is after. Using such a form, the system then attempts to construct a query by identifying the sources that provide the elements specified by the user, and connects the schemas of the sources selected using previously specified schema mappings. Of relevance to this problem is the proposal by Blunschi *et al.* [4], which considers indexing support for queries that combine keywords and structure and proposes several extensions to inverted lists to capture structure when it is present. In particular, it takes into account attribute labels, relationships between data items, hierarchies of schema elements, and synonyms among schema elements. We can also foresee the application of techniques taken from different areas in which the problem of search in semistructured or non structured data was already addressed [8,1,6,15]. In general, multiple structured queries are constructed from a set of keywords. The issue that needs to be addressed is, therefore, to identify the queries that closely meet user expectations.

*Usage and Improvement.* The results returned by each of the sources involved in a multi-domain search task are uncertain; this uncertainty is partly due to the fact that such results are generally obtained by matching a request with the content of the source in question using heuristics. The difficulty then lies in specifying a function whereby the results obtained by combining the sub-results retrieved from the sources involved in the search task can be ranked; this is a *ranking composition* problem [5]. Currently, ranking composition functions are typically manually specified, a task that can be difficult since it involves defining the global ranking of the query results taking into consideration the (possibly different) ranking criteria adopted by the underlying search sources.

A possible solution to the above issue can be borrowed from dataspace through pay-as-you-go development of ranking composition. The results returned

by evaluating a user query in dataspace are also uncertain; this uncertainty is partly due to the fact that the mappings used for populating the elements of the integration schema (against which the user queries are posed) are derived based on the results of matching heuristics [33] (see *mappings identification* dimensions). These mappings may not be manually debugged, but rather may be verified by seeking feedback from end users (see *improvement* dimensions). A similar approach can be adopted for specifying ranking composition in the context of multi-domain search tasks. For example, the user can specify that a given result should appear before another one. Using this kind of feedback, the system can then learn the ranking desired by the user. Which mechanism to use for learning the correct ranking is an open issue. Existing ranking methods and algorithms in the information retrieval literature are potentially relevant for this purpose [9].

## 5 Conclusions

Dataspace represent a vision for incremental refinement in data integration, in which the effort devoted to refining a dataspace can be balanced against the cost of obtaining higher quality integration. Comprehensive support for pay-as-you-go data integration might be expected to support different forms of refinement, where both the type and quantity of feedback sought are matched to the specific requirements of an application, user community or individual. Early proposals, however, provide rather limited exploration of the space of possibilities for incremental improvement. As the large number of dimensions in the classification shows, the decision space facing the designers of dataspace has many aspects. In this context, a common emphasis has been on reducing start-up costs, for example by supporting a *union* integration schema; such an approach provides syntactic consistency, but the extent to which the resulting dataspace can be said to “integrate” the participating sources is somewhat limited.

Although there is a considerable body of work outside dataspace to support activities such as schema matching or merging, early dataspace proposals have made fairly limited use of such techniques. Furthermore, there are no comparable results on automated refinement. As such, there is some way to go before the full range of dimensions associated with dataspace are associated with substantive results, and even where this is the case there will be considerable challenges composing these results to provide dataspace deployments that meet diverse user requirements. However, dataspace provide an overall vision that promises to enable the wider application of information integration techniques, by balancing the costs of integration activities with their benefits. The challenge of providing appropriate data integration at manageable cost seems to be of widespread relevance in widely different contexts, including personal, group, enterprise and web scale settings, acting over sources that provide computational services, structured data access and search. This chapter has sought to characterise the area, with a view to comparing the contributions to date, identifying topics for further investigation, and clarifying the space of issues of relevance to pay-as-you-go integration.

With respect to the interplay between search computing and dataspace, we note that techniques from search computing can be borrowed to address issues that arise within dataspace, and vice versa. In particular, search computing techniques can be used in dataspace when queries need to be evaluated using search sources. On the other hand, the pay-as-you-go dataspace philosophy can be used in search computing for incrementally defining ranking functions based on feedback supplied by end users.

## References

1. Amer-Yahia, S., Botev, C., Shanmugasundaram, J.: Textquery: a full-text search extension to xquery. In: WWW 2004: Proceedings of the 13th international conference on World Wide Web, pp. 583–594. ACM, New York (2004)
2. Atzeni, P., Cappellari, P., Torlone, R., Bernstein, P.A., Gianforme, G.: Model-independent schema translation. VLDB J. 17(6), 1347–1370 (2008)
3. Belhajjame, K., Paton, N.W., Embury, S.M., Fernandes, A.A., Hedeler, C.: Feedback-based annotation, selection and refinement of schema mappings for dataspace. In: EDBT (2010)
4. Blunschi, L., Dittrich, J.-P., Girard, O.R., Karakashian, S.K., Salles, M.A.V.: A dataspace odyssey: The imemex personal dataspace management system (demo). In: CIDR, pp. 114–119 (2007)
5. Braga, D., Ceri, S., Daniel, F., Martinenghi, D.: Optimization of multi-domain queries on the web. PVLDB 1(1), 562–573 (2008)
6. Cafarella, M.J., Etzioni, O.: A search engine for natural language applications. In: WWW 2005: Proceedings of the 14th international conference on World Wide Web, pp. 442–452. ACM, New York (2005)
7. Cafarella, M.J., Halevy, A.Y., Khoussainova, N.: Data integration for the relational web. PVLDB 2(1), 1090–1101 (2009)
8. Chakrabarti, S., Puniyani, K., Das, S.: Optimizing scoring functions and indexes for proximity search in type-annotated corpora. In: WWW 2006: Proceedings of the 15th international conference on World Wide Web, pp. 717–726. ACM, New York (2006)
9. Chaudhuri, S., Das, G., Hristidis, V., Weikum, G.: Probabilistic information retrieval approach for ranking of database query results. ACM Trans. Database Syst. 31(3), 1134–1168 (2006)
10. Dittrich, J.-P., Salles, M.A.V.: idm: A unified and versatile data model for personal dataspace management. In: VLDB 2006: 32nd International Conference on Very Large Data Bases, pp. 367–378. ACM, New York (2006)
11. Doan, A., Ramakrishnan, R., Chen, F., DeRose, P., Lee, Y., McCann, R., Sayyadian, M., Shen, W.: Community information management. IEEE Data Eng. Bull. 29(1), 64–72 (2006)
12. Dong, X., Halevy, A.Y.: A platform for personal information management and integration. In: CIDR 2005, pp. 119–130 (2005)
13. Dong, X., Halevy, A.Y., Yu, C.: Data integration with uncertainty. In: VLDB 2007: 33rd International Conference on Very Large Data Bases, pp. 687–698 (2007)
14. Dong, X.L., Halevy, A.Y., Yu, C.: Data integration with uncertainty. VLDB J. 18(2), 469–500 (2009)

15. Florescu, D., Kossmann, D., Manolescu, I.: Integrating keyword search into xml query processing. In: Proceedings of the 9th international World Wide Web conference on Computer networks: the international journal of computer and telecommunications networking, pp. 119–135. North-Holland Publishing Co., Amsterdam (2000)
16. Franklin, M., Halevy, A., Maier, D.: From databases to daspases: a new abstraction for information management. SIGMOD Record 34(4), 27–33 (2005)
17. Haas, L., Lin, E., Roth, M.: Data integration through database federation. IBM Systems Journal 41(4), 578–596 (2002)
18. Halevy, A., Franklin, M., Maier, D.: Principles of daspace systems. In: PODS 2006: Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, pp. 1–9. ACM, New York (2006)
19. Hedeler, C., Belhajjame, K., Fernandes, A.A.A., Embury, S.M., Paton, N.W.: Dimensions of daspases. In: Sexton, A.P. (ed.) BNCOD 2009. LNCS, vol. 5588, pp. 55–66. Springer, Heidelberg (2009)
20. Howe, B., Maier, D., Rayner, N., Rucker, J.: Quarrying daspases: Schemaless profiling of unfamiliar information sources. In: ICDE Workshops, pp. 270–277. IEEE Computer Society, Los Alamitos (2008)
21. Ives, Z.G., Green, T.J., Karvounarakis, G., Taylor, N.E., Tannen, V., Talukdar, P.P., Jacob, M., Pereira, F.: The orchestra collaborative data sharing system. SIGMOD Record 37(3), 26–32 (2008)
22. Ives, Z.G., Knoblock, C.A., Minton, S., Jacob, M., Talukdar, P.P., Tuchinda, R., Ambite, J.L., Muslea, M., Gazen, C.: Interactive data integration through smart copy & paste. In: CIDR (2009)
23. Jeffery, S.R., Franklin, M.J., Halevy, A.Y.: Pay-as-you-go user feedback for daspace systems. In: SIGMOD 2008: Proceedings of the 2008 ACM SIGMOD international conference on Management of data, pp. 847–860. ACM, New York (2008)
24. Leser, U., Naumann, F.: (almost) hands-off information integration for the life sciences. In: Conf. on Innovative Database Research (CIDR), pp. 131–143 (2005)
25. Llu, J., Dong, X., Halevy, A.: Answering structured queries on unstructured data. In: WebDB 2006, pp. 25–30 (2006)
26. Madhavan, J., Bernstein, P.A., Doan, A., Halevy, A.: Corpus-based schema matching. In: International Conference on Data Engineering (ICDE 2005), pp. 57–68 (2005)
27. Madhavan, J., Cohen, S., Dong, X.L., Halevy, A.Y., Jeffery, S.R., Ko, D., Yu, C.: Web-scale data integration: You can afford to pay as you go. In: CIDR 2007: Third Biennial Conference on Innovative Data Systems Research, pp. 342–350 (2007)
28. Madhavan, J., Cohen, S., Dong, X.L., Halevy, A.Y., Jeffery, S.R., Ko, D., Yu, C.: Web-scale data integration: You can afford to pay as you go. In: CIDR, pp. 342–350 (2007)
29. McCann, R., Shen, W., Doan, A.: Matching schemas in online communities: A web 2.0 approach. In: ICDE, pp. 110–119 (2008)
30. Miller, R.J., Hernández, M.A., Haas, L.M., Yan, L., Ho, C.T.H., Fagin, R., Popa, L.: The clio project: managing heterogeneity. SIGMOD Record 30(1), 78–83 (2001)
31. Pottinger, R., Bernstein, P.A.: Schema merging and mapping creation for relational sources. In: EDBT, pp. 73–84 (2008)
32. Rahm, E., Bernstein, P.A.: A survey of approaches to automatic schema matching. VLDB Journal: Very Large Data Bases 10(4), 334–350 (2001)
33. Rahm, E., Bernstein, P.A.: A survey of approaches to automatic schema matching. VLDB J. 10(4), 334–350 (2001)

34. Salles, M.A.V., Dittrich, J.-P., Karakashian, S.K., Girard, O.R., Blunski, L.: itrails: Pay-as-you-go information integration in dataspace. In: VLDB 2007: 33rd International Conference on Very Large Data Bases, pp. 663–674. ACM, New York (2007)
35. Sarma, A.D., Dong, X., Halevy, A.: Bootstrapping pay-as-you-go data integration systems. In: SIGMOD 2008: Proceedings of the 2008 ACM SIGMOD international conference on Management of data, pp. 861–874. ACM, New York (2008)
36. Sarma, A.D., Dong, X.L., Halevy, A.Y.: Data modeling in dataspace support platforms. In: Borgida, A.T., Chaudhri, V.K., Giorgini, P., Yu, E.S. (eds.) Conceptual Modeling: Foundations and Applications. LNCS, vol. 5600, pp. 122–138. Springer, Heidelberg (2009)
37. Talukdar, P.P., Jacob, M., Mehmood, M.S., Crammer, K., Ives, Z.G., Pereira, F., Guha, S.: Learning to create data-integrating queries. PVLDB 1(1), 785–796 (2008)
38. Tatemura, J., Chen, S., Liao, F., Po, O., Candan, K.S., Agrawal, D.: Uqbe: uncertain query by example for web service mashup. In: SIGMOD Conference, pp. 1275–1280 (2008)

# Chapter 8:

## Multimedia and Multimodal Information Retrieval

Alessandro Bozzon and Piero Fraternali

Dipartimento di Elettronica e Informazione, Politecnico di Milano,  
Piazza Leonardo da Vinci 32, 20133 Milano, Italy  
{alessandro.bozzon, piero.fraternali}@polimi.it

**Abstract.** The Web is progressively becoming a multimedia content delivery platform. This trend poses severe challenges to the information retrieval theories, techniques and tools. This chapter defines the problem of multimedia information retrieval with its challenges and application areas, overviews its major technical issues, proposes a reference architecture unifying the aspects of content processing and querying, exemplifies a next-generation platform for multimedia search, and concludes by showing the close ties between multi-domain search investigated in Search Computing and multimodal/multimedia search.

**Keywords:** multimedia information retrieval, digital signal processing, video search engines, multi-modal query interfaces.

### 1 Introduction

The growth of digital content has reached impressive rates in the last decade, fuelled by the advent of the so-called “Web 2.0” and the emergence of user-generated content. At the same time, the convergence of the fixed-network Web, mobile access, and digital television has boosted the production and consumption of audio-visual materials, making the Web a truly multimedia platform.

This trend challenges search as we know it today, due to the more complex nature of multimedia with respect to text, in all the phases of the search process: from the expression of the user’s information need to the indexing of content and the processing of queries by search engines.

This Chapter gives a concise overview of Multimedia Information Retrieval (MIR), the long-standing discipline at the base of audio-visual search engines, and connects the research challenges in this area to the objectives and research goals of Search Computing.

MIR amplifies many of the research problems at the base of search over textual data. The grand challenge of MIR is bridging the gap between queries and content: the former are either expressed by keywords, like in text search engines, or, by extension, with non-textual samples (e.g., an image or a piece of music). Unlike in text search engines, where the query has the same format of content and can be matched almost directly to it, query processing in MIR must fill an enormous gap. To understand if an image, video or piece of music is relevant to some keyword, it is necessary to extract the hidden knowledge buried inside the aural and visual

resources, a multi-sensorial recognition problem that in nature living organisms took quite a long time to solve.

Not surprisingly, MIR research revolves around the problem of extracting, organizing and making available for querying the knowledge present inside media assets. This problem is far from being solved in general, but many effective techniques have been devised for special cases, typically for the extraction of specific “features” from specific non-textual resources. Applications like mood classification and similarity matching, face recognition, video optical character recognition are examples of these techniques, already deployed in commercial multimedia search solutions.

Since giving the full account of MIR research goes beyond the limits of this Chapter, we have organized the illustration so as to give a flavor of the essential themes. After exemplifying the numerous applications that motivate the growing interest in MIR (Section 1.1), Section 2 overviews the principal research topics in the development of a MIR solution: from the acquisition of content (Section 2.1), to its normalization for the purpose of processing (Section 2.2), to the extraction of the features useful for searching and their organization by means of suitable indexes (Section 2.3), to the languages and algorithms for processing queries (Section 2.4), to the problem of presenting search results (Section 2.5).

The variety of MIR solutions available can be abstracted by a common architecture, which is the subject of Section 3; a MIR system can be seen as an infrastructure for governing two main processes: the *content process*, treated in Section 3.1, comprises all the steps necessary to extract indexable features (called *metadata*) from multimedia elements; the query process, overviewed in Section 3.2, includes all the steps for executing a user’s query.

The link between the content process and the query process is represented by *metadata*, which encode the knowledge that the MIR system is able to extract from the media assets, index and use for answering queries. Given that no single universal standard still exists for MIR metadata, Section 4 overviews some of the most popular formats that have been proposed in different application domains. How to extract such metadata from audiovisual data is the subject of Section 5, which presents a bird’s eyes view of some feature extraction approaches for audio, image, and video content. This is the area where current research is most active, because the problem of understanding the content of non-textual data is far from being solved in a general way. In Section 6, we also provide an overview of the different query languages used in MIR, which go from simple keyword queries to structured languages.

To make the Chapter more concrete, Section 7 mentions a number of research and commercial MIR systems, where the architecture and techniques described in the preceding Sections have been put to work.

We conclude the Chapter with an outlook (in Section 8) of what lessons can be mutually learnt by researchers in MIR and Search Computing. As in MIR, Search Computing relies on a well balanced mix of offline content preparation (the wrapping and registration of heterogeneous data sources) and smart query processing; moreover, the presentation of multimedia search results requires smart solutions for easing the interpretation of complex results sets, which exhibit sophisticated internal structure, and a spatial as well as temporal distribution. MIR systems, pioneers of a search technology that goes beyond textual Web pages, may be an interesting source



of inspiration for the multi-domain content integration, query processing, and result presentation challenges that Search Computing is facing.

### 1.1 Motivations, Requirements and Applications of Multimedia Search

“Finding the title and author of a song recorded with one’s mobile in a crowded disco”; “Locating news clips containing interviews to President Obama and accessing the exact point where the Health Insurance Reform is discussed”; “Finding a song matching in mood the images to be placed in a slideshow”. These are only a few examples of what multimedia information retrieval is about: satisfying a user’s information need that spans across multiple media, which can itself be expressed using more than one medium.

The requirements of a MIR application bring to the extreme or go beyond the problems faced in classical text information retrieval [37]:

- **Opacity of Content:** whereas in text IR the query and the content use the same medium, MIR content is opaque, in the sense that the knowledge necessary to verify if an item is relevant to a user’s query is deeply embedded in it and must be extracted by means of a complex pre-processing (e.g., extracting speech transcriptions from a video).
- **Query Formulation Paradigm:** as for traditional search engines, keywords may not be the only way of seeking for information: for instance, queries can be expressed by analogy, submitting a sample of content “similar” to what the user is searching for. In MIR, content samples used as queries can be as complex as an image, a piece of music, or even a video fragment.
- **Relevance Computation:** in text search, relevance of documents to the user’s query is computed as the similarity degree between the vectors of *words* appearing in the document and in the query (modulo lexical transformations). In MIR, the comparison must be done on a much wider variety of features, characteristic not only of the specific medium in which the content and the query are expressed, but even of the application domain (e.g., two audio files can be deemed similar in a music similarity search context, but dissimilar in a topic-based search application).

MIR applications requirements have been extensively addressed in the last three decades, both in the industrial and academic fields. As a consequence, MIR is now a consolidated discipline, adopted into a wide variety of domains [41], including:

- Architecture, real estate, and interior design (e.g., searching for ideas).
- Broadcast media selection (e.g., radio channel [58], TV channel).
- Cultural services (history museums [11], art galleries, etc.).
- Digital libraries (e.g., image catalogue [69], musical dictionary, bio-medical imaging catalogues [4], film, video and radio archives [52]).
- E-Commerce (e.g., personalized advertising, on-line catalogues [53]).
- Education (e.g., repositories of multimedia courses, multimedia search for support material).
- Home Entertainment (e.g., systems for the management of personal multimedia collections [27], including manipulation of content, e.g. home video editing [2], searching a game, karaoke).

- Investigation (e.g., human characteristics recognition [22], forensics [40]).
- Journalism (e.g. searching speeches of a certain politician [25] using his name, his voice or his face [23]).
- Multimedia directory services (e.g. yellow pages, Tourist information, Geographical information systems).
- Multimedia editing (e.g., electronic news service [16], media authoring).
- Remote sensing (e.g., cartography, ecology [81], natural resources management).
- Social (e.g. dating services, podcast [54] [56]).
- Surveillance (e.g., traffic control, surface transportation, non-destructive testing in hostile environments).

## 2 Challenges of Multimedia Information Retrieval

Multimedia search engines and their applications operate on a very heterogeneous spectrum of content, ranging from home-made content created by users to high value premium productions, like feature film video. The quality of content largely determines the kind of processing that is possible for extracting information and the kind of queries that can be answered. This Section overviews the main challenges in the design of a MIR solution, by following the lifecycle of multimedia content, from its entrance into the system (acquisition), to its preparation for analysis (normalization), to the extraction of metadata necessary for building the search engine indexes (indexing), to the processing of a user's query (querying) and, finally, to the presentation of results (browsing).

### 2.1 Challenge 1: Content Acquisition

In text search engines, content comes either from a closed collection (as, e.g., in a digital library) or is crawled from the open Web. In MIR, multimedia content can be acquired in a way similar to document acquisition:

- By crawling the Web or local media repositories.
- By user's contribution or syndicated contribution from content aggregators.

Additionally, multimedia content can also come directly from production devices directly connected to the system, such as scanners, digital cameras, smartphones, or broadcast capture devices (e.g., from air/cable/satellite broadcast, IPTV, Internet TV multicast, etc.).

Besides the heterogeneity of acquisition sources and protocols, also the size of media files make the content ingestion task more complicated, e.g., because the probability of download failures increases, the cost of storing duplicates or near duplicates becomes less affordable, and the presence of DRM issues on the downloaded content is more frequent.

As for textual data, but even more critical in the case of audiovisual content, is the capability of the content ingestion subsystem to preserve or even enhance the intrinsic quality of the downloaded digital assets, e.g., by acquiring them at the best

resolution possible, given the bandwidth limitations, and preserving all the available *metadata* associated with them.

Metadata are textual descriptions that accompany a content element; they can range in quantity and quality, from no description (e.g., Webcam content) to multilingual data (e.g., closed captions and production metadata of motion pictures). Metadata can be found:

- Embedded within content (e.g., video close captions or Exchangeable image file format (EXIF) data embedded in images).
- In surrounding Web pages or links (e.g., HTML content, link anchors, etc).
- In domain-specific databases (e.g., IMDB [72] for feature films).
- In ontologies (e.g., like those listed in the DAML Ontology Library [71]).

The challenge here is building scalable and intelligent content acquisition systems, which could ingest content exploiting different communication protocols and acquisition devices, decide the optimal resolution in case alternative representations are available, detect and discard duplicates as early as possible, respect DRM issues, and enrich the raw media asset with the maximum amount of metadata that could be found inside or around it.

## 2.2 Challenge 2: Content Normalization

In textual search engines, context is subjected to a pipeline of operations for preparing it to be indexed [3]; such pre-processing includes parsing, tokenization, lemmatization, and stemming. With text, the elements of the index are of the same nature of the constitutive elements of content: *words*. Multimedia content needs a more sophisticated pre-processing phase, because the elements to be indexed (called “features” or “annotations”) are numerical and textual metadata that need to be extracted from raw content by means of complex algorithms.

The processing pipeline for multimedia data is therefore longer than in text search engines, and can be roughly divided in two macro steps: content normalization (treated in this Section) and content analysis (treated in the next Section).

Due to the variety of multimedia encoding formats, prior to processing content for metadata extraction, it is necessary to submit it to a normalization step, with a twofold purpose: 1) translating the source media items represented in different native formats into a common representation format (e.g., MPEG4 [49] for video files), for easing the development and execution of the metadata extraction algorithms; 2) producing alternative variants of native content items, e.g., to provide freebies (free sample copies) of copyrighted elements or low resolution copies for distribution on mobile or low-bandwidth delivery channels (e.g., making a 3GP version [70] of video files for mobile phone fruition). The challenge here is to devise the best encoding format for addressing the needs of analysis algorithm and easing the delivery of content at variable quality, without exploding the number of versions of the same item to be stored in the search engine.

## 2.3 Challenge 3: Content Analysis and Indexing

After the normalization step, a multimedia collection has to be processed in order to make the knowledge embedded in it available for querying, which requires building

the internal indexes of the search engine. Indexes are a concise representation of the content of an object collection, constructed out of the features extracted from it; the features used to build the indexes must be both sufficiently representative of the content and compact to optimize storage and retrieval.

Features are traditionally grouped into two categories:

- *Low level features*: concisely describe physical or perceptual properties of a media element (e.g., the colour or edge histogram of an image).
- *High level features*: domain concepts characterizing the content (e.g., extracted objects and their properties, geographical references, etc.).

As in text, where the retrieved keywords can be highlighted in the source document, also in MIR there is the need of locating the occurrences of matches between the user's query and the content. Such requirement implies that features must be extracted from a time continuous medium, and that the coordinates in space and time of their occurrence must be extracted as well (e.g., the time stamp at which a word occurs in a speech audio file, the bounding-box where an object is located in an image, or both pieces of information to denote the occurrence of an object in a video).

Feature detection may even require a change of medium with respect to the original file, e.g., the speech-to-text transcription.

Content analysis and indexing are the prominent research problem of MIR, as the quality of the search engine depends on the precision at which the extracted metadata describe the content of a media asset: after introducing the global scheme of the content analysis process in Section 3.1, we devote Section 4 to the various ways in which features (also called metadata) can be represented and Section 5 to the algorithms for computing them.

## 2.4 Challenge 4: Content Querying

Text IR starts from a user's query, formulated as a set of keywords, possibly connected by logical operators (AND, OR, NOT). The semantics of query processing is text similarity: both the text files and the query are represented into a common logical model (e.g., the word vector model [64]), which supports some form of similarity measure (e.g., cosine similarity between word vectors).

In MIR, the expression of the user's information need allows for alternative query representation formats and matching semantics. Examples of queries can be:

- *Textual*: one or more keywords, to be matched against textual metadata extracted from multimedia content.
- *Mono-media*: a content sample in a single media (e.g., an image, a piece of audio) to be matched against an item of the same kind (e.g., query by music or image similarity, query by humming) or of a different medium (e.g., finding the movies whose soundtrack is similar to an input audio file).
- *Multi-media*: a content sample in a composite medium, e.g., a video file to be matched using audio similarity, image similarity, or a combination of both.

Accepting in input queries expressed by means of non-textual samples requires real-time content analysis capability, which poses severe scalability requirements on MIR architectures. Another implication of non-textual queries is the need for the MIR

architecture to coordinate query processing across multiple dedicated search engines: for example, an image similarity query may be responded by coordinating an image similarity search engine specialized in low-level features matching and a text search engine, matching high-level concepts extracted from the query (e.g., object names, music gender, etc).

The grand challenge of MIR query processing is in part the same as for textual IR: retrieving the media objects more relevant to the user's query with high precision and recall. MIR adds the specific problem of content-based queries, which demand suitable architectures for analysing a query content sample on the fly and matching its features to those stored in the indexes. We devote Section 3.2 to a brief overview of the query process.

## 2.5 Challenge 5: Content Browsing

Unlike data retrieval queries (such as SQL or XPATH queries), IR queries are approximate and thus results are presented in order of relevance, and often in a number that exceeds the user's possibility of selection. Typically, a text search engine summarizes and pages the ranked results, so that the user can quickly understand the most relevant items.

In MIR applications, understanding if a content element is relevant poses additional challenges. On one side, content summarization is still an open problem [5]: for example, a video may be summarized in several alternative ways: by means of textual metadata, with a selection of key frames, with a preview (e.g., the first 10

The screenshot displays the 'Result Details' page for a video titled 'Aransas and Matagorda Island National Wildlife Refuges (2005)'. The video player shows a scene with several white birds in a natural setting. To the right, the 'Video passages' section lists various image-based annotations with their corresponding percentages:

- (image) grass (80%)
- (image) vegetation (58%)
- (image) greenery (56%)
- (image) grass (40%)
- (image) birds (33%)
- (image) albatross (33%)
- (image) wings (33%)
- (keyframe)
- (image) vegetation (29%)
- (image)

At the bottom of the interface, there are two horizontal time bars: 'What we see' (visual time bar) and 'What we hear' (aural time bar), both showing segmented bars representing different content elements over time.

Fig. 1. Visual and aural time bars in the interface of the PHAROS search platform [10]

seconds), or even by means of another correlated item (e.g., the free trailer of a copyrighted feature film). The interface must also permit users to quickly inspect continuous media and locate the exact point where a match has occurred. This can be done in many ways, e.g., by means of annotated time bars that permit one to jump into a video where a match occurs, with VCR-like commands, and so on.

Figure 1 shows a portion of the user interface of the PHAROS multimedia search platform [10] for accessing video results of a query: two time bars (labelled “what we hear”, “what we see”) allow one to locate the instant where the matches for a query occur in the video frames and in the audio, inspect the metadata that support the match, and jump directly to the point of interest.

The challenge of MIR interfaces is devising effective renditions (visual, but also aural) that could convey both the global characteristics of the result set (e.g., the similarity distribution across a result collection) and the local features of an individual result item that justify the query match.

### 3 The MIR Architecture

The architecture of a MIR system [9] can be described as a platform for composing, verifying, and executing search processes, defined as complex workflows made of atomic blocks, called search services, as illustrated in Figure 2. At the core of the architecture there is a *Process Execution Engine* which is a runtime environment, optimized for the scalable enactment of data-intensive and computation-intensive workflows made of search services. A *search service* is a wrapper for any software component that embodies functionality relevant to a MIR solution.

The most important categories of MIR workflows are *Content Processes*, which have the objective of acquiring multimedia content from external sources (e.g. from the user or a from video portal) and extracting features from it; and the *Query Processes*, which have the objective of acquiring a user’s information need and computing the best possible answer to it. Accordingly, the most important categories of search services are *content services*, which embody functionality relevant to content acquisition, analysis, enrichment, and adaptation; and *query services*, which implements all the steps for answering a query and computing the ranked list of results.

Examples of content services can be: algorithms for extracting knowledge from media elements, transducers for modifying the encoding format of media files; examples of query services, instead, are: query disambiguation services for inferring the meaning of ambiguous information needs, or social network analysis services for inferring the preferences of a user and personalizing the results of a user’s query.<sup>1</sup>

---

<sup>1</sup> In Figure 2 metadata are given in output to the content owner. They enrich the processed content and thus increase its value, and thus can be used by the content owner for publishing purposes or for building a separate query processing solution.

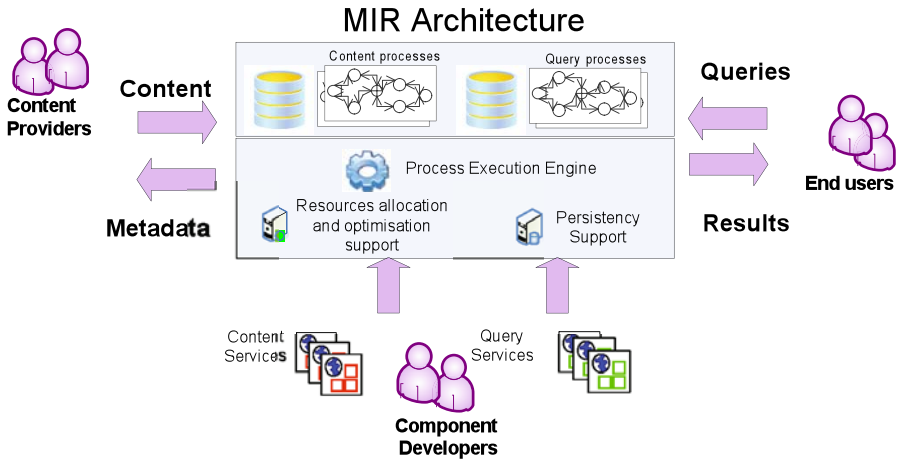


Fig. 2. Reference architecture of a MIR system

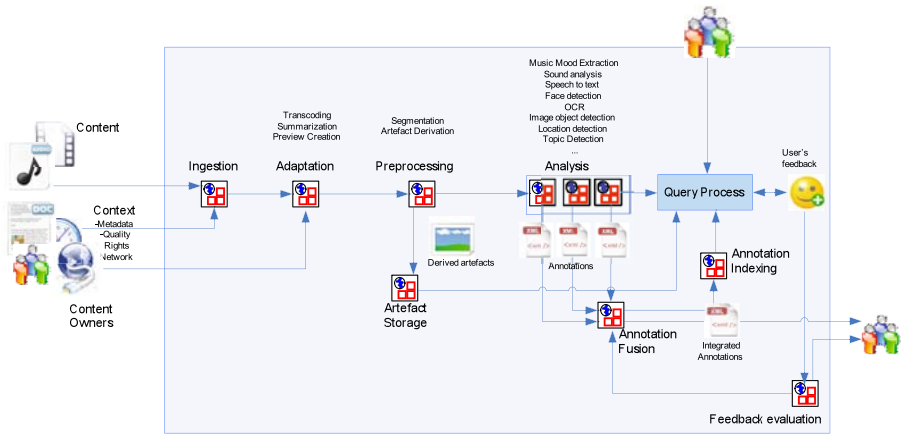
### 3.1 The Content Process

A content process (as the one schematized in Figure 3) aims at gathering multimedia content and at elaborating it to make it ready for information retrieval. A MIR platform may host multiple content processes, as required for elaborating content of different nature, in different domains, for different access devices, for different business goals, etc.

The input to the process is twofold:

- Multimedia Content (image, audio, video).
- Information about the content, which may include *publication metadata* (HTML, podcast [55], RSS [62], MediaRSS [45], MPEG7, etc), *quality information* (encoding, user's rating, owner's ratings, classification data), *access rights* (DRM data, licensing, user's subscriptions), and *network information* (type and capacity of the link between the MIR platform and the content source site - e.g., access can be local disk-based, remote though a LAN/SAN, a fixed WAN, a wireless WAN, etc).

The output of the process is the textual representation of the metadata that capture the knowledge automatically extracted from the multimedia content via content processing operations. The calculated metadata are integrated with the metadata gathered by the content acquisition system (shown as an input in Figure 3), which are typically added to the content manually by the owner or by the Web users (e.g., as tags, comments, closed captions, and so on). Section 4 and Section 5 respectively provide a discussion on the state of the art of metadata vocabularies and analysis techniques for extracting metadata from multimedia assets.



**Fig. 3.** Example of a MIR Content Process

A content process can be designed so as to dynamically adapt to the external context, e.g., as follows:

- By analyzing the content metadata (e.g., manual annotations) to dynamically decide the specific analysis operators to apply to a media element (e.g., if the collection denotes indoor content, a heuristic rule may decide to skip the execution of outdoor object detection).
- By analyzing the access rights metadata to decide the derived artefacts to extract (e.g., if content has limited access, it may be summarized in a freebie version for preview)
- By analyzing the geographical region where the content comes from (e.g., inferring the location of the publisher may allow the process to apply better heuristic rules for detecting the language of the speech and call the proper speech-to-text transcription module).
- By understanding the content delivery modality (e.g., a real-time stream of a live event may be indexed with a faster, even if less precise, process for reducing the time-to-search delay interval).

### 3.2 The Query Process

A MIR Query process (like the one schematized in Figure 4) accepts in input information need and formulates the best possible answer from the content indexed in the MIR platform.

The input of the query process is an information need, which can be a keyword or a content sample. The output is a result set, which contains information on the objects (typically content elements) that match the input query. The description of the objects in the result set can be enriched with metadata coming from sources external to the MIR platform (e.g., additional metadata on a movie taken from IMDB, or a map showing the position of the object taken from a Geographical Information System).



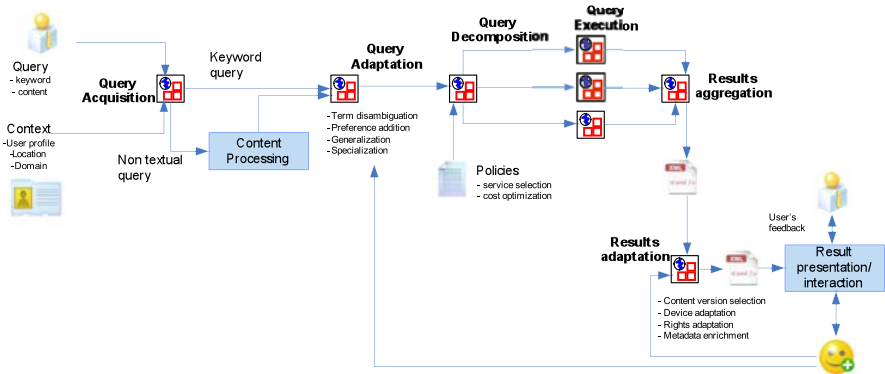


Fig. 4. Example of a MIR Query Process

A collateral source of input is the query context, which expresses additional circumstances about the information need, often implicit. Well-known examples of query context are: user preferences, past users' queries and their responses, access device, location, access rights, and so on. The query context is used to adapt the query process: for instance, it can be used to expand the original information need of the user with additional keywords reflecting her preferences, to disambiguate a query term based on the application domain where the query process is embedded, or to provide the best shape of results for the current user.

Queries are classified as **mono-modal**, if they are represented in a single medium (e.g., a text keyword, a music fragment, an image) or **multi-modal**, if they are represented in more than one medium (e.g., a keyword AND an image). As for Search Computing, also in MIR queries can be classified as **mono-domain**, if they are addressed to a single search engine (e.g., a general purpose image search engine like Google Images [26] or a special purpose search service as Empora [20] garments search), or **multi-domain**, if they target different, independent search services (e.g., a face search service like Facesearch [23] and a video search service like Blinkx [7]). Table 1 exemplifies the domain and mode classification of queries.

Table 1. Examples of Mono/Multi modal, Mono/Multi domain queries in a MIR system

	Mono Domain	Multi Domain
Mono Modal	Find all the results that match a given keyword; Find all the images similar to a given image.	Find theatres playing movies acted by an actor having the voice similar to a given one.
Multi Modal	Find all videos that contain a given keyword and that contain a person with a face similar to a given one.	Find all CDs in Amazon with a cover similar to a given image.

## 4 Metadata

The content process produces a description of the knowledge extracted from the media assets, possibly integrated with information gathered during the content

acquisition phase. This articulated knowledge must be represented by means of a suitable formalism: the current state of the practice in content management presents a number of metadata vocabularies dealing with the description of multimedia content [24]. Many vocabularies allow the description of high-level (e.g., title, description) or low-level features (e.g., colour histogram, file format), while some enable the representation of administrative information (e.g., copyright management, authors, date). In a MIR system, the adoption of a specific metadata vocabulary depends on its intended usage, especially for what concerns the type of content to describe. In the following we illustrate a few relevant and diverse examples that cover the main metadata format categories.

**MPEG-7** [42] is an XML vocabulary that represents the attempt from ISO to standardize a core set of audio-visual features and structures of descriptors and their (spatial/temporal) relationships. By trying to abstract from all the possible application domains, MPEG-7 results in an elaborate and complex standard that merges both high-level and low-level features, with multiple ways of structuring annotations. MPEG7 is also extensible, so to allow the definition of application-based or domain-based metadata.

**Dublin Core** [19] is a 15-element metadata vocabulary (created by domain experts in the field of digital libraries) intended to facilitate discovery of electronic resources, with no fundamental restriction on the resource type. Dublin Core holds just a small set of high-level metadata and relations (e.g. title, creator, language, etc...), but its simplicity made it a common annotation scheme across different domains. It can be encoded using different concrete syntaxes, e.g., in plain text, XML or RDF.

**MXF** (Material Exchange Format) [17] is an open file format that wraps video, audio, and other bit streams (called "essences"), aimed at the interchange of audio-visual material, along with associated data and metadata, in devices ranging from cameras and video recorders to computer systems for various applications used in the television production chain. MXF metadata address both high-level and administrative information, like the file structure, key words or titles, subtitles, editing notes, location, etc. Though it offers a complete vocabulary, MXF has been intended primarily as an exchange format for audio and video rather than a description format for metadata storage and retrieval.

**Exchangeable Image File Format (EXIF)** [31] is a vocabulary adopted by digital camera manufacturers to encode high-level metadata like date and time information, the image title and description, the camera settings (e.g., exposure time, flash), the image data structure (e.g., height, width, resolution), a preview thumbnail, etc. By being embedded in picture raw contents, EXIF metadata is now a de-facto standard for image management software; to support extensibility, EXIF enables the definition of custom, manufacturer-dependent additional terms.

**ID3** [32] is a tagging system that enriches audio files by embedding metadata information. ID3 includes a big set of high-level (such as title, artist, album, genre) and administrative information (e.g. the license, ownership, recording dates), but a very small set of low-level information (e.g. BPM). ID3 is a worldwide standard for audio metadata, adopted in a wide set of applications and hardware devices. However, ID3 vocabulary is fixed, thus hindering its extensibility and usage as format for low-level features.

Other examples of multimedia-specific metadata formats are SMEF [67] for video, IPTC [34] for images, and MusicXML [61] for music. In addition, several communities created some domain-specific vocabularies like LSCOM [39] for visual concepts, IEEE LOM [36] for educational resources, and NewsML [34] for news objects.

## 5 Techniques for Content Processing

The *content process* described in Figure 3 aims at creating a representation of the multimedia collection suitable for indexing and retrieval purposes. The techniques applied for analysing content are application dependent, and relate both with the nature of the processed items and with the aim of the applications. The content process is not exclusive of MIR, but also applies classical text-based IR systems. The processing of text is a well-understood activity which embodies a standard sequence of operations (language detection, spell checking, correction and variant resolution, lemmatization, and stop-word removal), which convert documents into a canonical format for a more efficient indexation [3].

MIR systems deal with more complex media formats, like audio, video and images, and therefore require a more articulated analysis process to produce the metadata needed for indexing. In essence, a MIR content process can be seen as an acyclic graph of operators, in which each operator extracts different features from a media item, possibly with the help of the metadata previously extracted by other already executed operators. The various operators embody diverse algorithms for content analysis and feature extraction, which are the subjects of the research challenges briefly introduced in Section 2.3.

The operations that constitute the MIR content process can be roughly classified in three macro categories: *transformation*, *feature extraction*, and *classification*, based on the stage at which they occur in the analysis process and on the abstraction level of the information they extract from the raw content:

- **Transformation:** this kind of operation converts the format of media items, for making the subsequent analysis steps more efficient or effective. For instance, a video transformer can modify an MPEG2 movie file to a format more suitable for the adopted analysis technologies (e.g., MPEG); likewise, an audio converter can transform music tracks encoded in MP3 to WAV, to eliminate compression and make content analysis simpler and more accurate.
- **Feature Extraction:** calculates low-level representations of media contents, i.e. feature vectors, in order to derive a compact, yet descriptive, representation of a pattern of interest [14]. Such representation can be used to enable content based search, or as input for classification tasks. Examples of visual features for images are *colour*, *texture*, *shape*, etc. [28]; examples of aural features for music contents are *loudness*, *pitch*, *tone (brightness and bandwidth)*, *Mel-filtered Cepstral Coefficients*, etc. [76].
- **Classification:** assigns conceptual labels to content elements by analyzing their raw features; the techniques required to perform this operations are commonly known as machine learning. For instance, an image classifier can assign to image files annotations expressing the subject of the pictures (e.g., mountains, city, sky, sea, people, etc.), while an audio file can be analyzed in order to discriminate segments containing speech from the ones containing music.

**Table 2.** Content analysis techniques in MIR systems

Audio Analysis	Image Analysis	Video Analysis
<i>Audio segmentation</i> [44]: to split audio track according to the nature of its content. For instance, a file can be segment according to the presence of noise, music, speech, etc.	<i>Semantic Concept extraction</i> [38]: the process of associating high-level concepts (like <i>sky</i> , <i>ground</i> , <i>water</i> , <i>buildings</i> , etc.) to pictures.	<i>Scene detection</i> [59]: detection of scenes in a video clip; a scene is one of the subdivisions of a play in which the setting is fixed, or that presents continuous action in one place [60].
<i>Audio event identification</i> [57]: to identify the presence of events like gunshots and scream in an audio track.	<i>Optical character recognition</i> [6]: to translate <i>images</i> of handwritten, typewritten or printed text into an editable text.	<i>Video text detection and segmentation</i> [50]: to detect and segment text in videos in order to apply image OCR techniques.
<i>Music genre (mood) identification</i> [12] [46]: to identify the genre (e.g., rock, pop, jazz, etc.) or the mood of a song.	<i>Face recognition and identification</i> [75] [82]: to recognize the presence of a human face in an image, possibly identifying its owner.	<i>Video summarization</i> [5]: to create a shorter version of a video by picking important segments from the original.
<i>Speech recognition</i> [21]: to convert words spoken in an audio file into text. Speech recognition is often associated with Speaker identification [51], that is to assign an input speech signal to one person of a known <i>group</i>	<i>Object detection and identification</i> [80]: to detect and possibly identify the presence of a known object in the picture.	<i>Shot detection</i> [15]: detection of transitions between shots. Often shot detection is performed by means of Keyframe segmentation [13] algorithms that segment a video track according to the key frames produced by the compression algorithm.

Arbitrary combinations of transformation, feature extraction, and classification operations can result in several analysis algorithms. Table 2 presents a list of 14 typical audio, image, and analysis techniques; the list is not intended to be complete, but rather to give a glimpse on the analysis capabilities currently available for MIR systems. To provide the reader with a hook to the recent advancements in the respective fields, each analysis technique is referenced with a recent survey on the topic. The techniques shown in Table 2 can be used in isolation, to extract different features from an item. Since the corresponding algorithms are probabilistic, each extracted feature is associated with a confidence value that denotes the probability that item X contain feature Y. To increase the confidence in the detection, different analysis techniques can be used jointly to reinforce each other. Using the example of the movie file, the fact that in a single scene both the face and the voice of a person are identified as belonging to an actor “X” can be considered as a correlated event, so to describe the scene as “scene where actor X appears” with a high confidence. The cross reinforcement of analysis techniques is called *annotation fusion*: multiple features extracted from media are fused together to yield more robust classification detection [43]. For instance, multiple content segmentation techniques (e.g., shot detection and speaker’s turn segmentation) can be combined in order to achieve better

video splitting; voice identification and face identification techniques can be fused in order to obtain better person identification. Typically, the use of multiple techniques simultaneously can be computationally expensive, thus limiting this solution to such domains where accuracy in the content descriptions is more important than indexing speed.

## 6 Examples of MIR Query Languages

In MIR, a user's query is matched against the representation of content provided by one (or more) of the metadata formats described in Section 4. Given such a variety of data representations, there is not a standardized query language for MIR systems, as every retrieval framework provides its own proprietary solution. For such a reason, several proposals for a unified MIR query language have emerged in the last years, and this Section will provide an overview.

Given that multimedia objects are usually described textually, a natural choice for the query language is exploiting mature text retrieval techniques: for instance, free-text or keyword-based search, context queries, Boolean queries, pattern queries [3], or faceted queries [63].

Even if based on a conventional IR query languages, though, a query language for MIR must comply with additional requirements typical of aural and visual media types or of specific application domains [29]:

- *Schema independence*: given the multitude of metadata representation formats, a query language should not rely on a specific schema.
- *Arbitrary search scope granularity*: the query language must allow search of information both in the whole media object and in chunks thereof.
- *Media objects as query conditions*: a MIR query language should support content-based queries, in one of two ways: 1) providing a media object to use as a query condition and the information about the algorithm to use for its on-the-fly analysis; 2) providing a set of previously calculated low-level features to use as a query condition.
- *Arbitrary similarity measure*: the query language should enable the flexible representation of arbitrary ranking functions, so to suite application-specific needs.

The last two decades have witnessed to a lot of efforts in the definition of more expressive and structured query languages, designed specifically for multimedia retrieval. Among the most recent efforts, POQL<sup>MM</sup> [30], is a general purpose query language for object oriented multimedia databases exposing arbitrary data schema. MuSQL [78] is a music structured query language, composed of a schema definition sub-language and a data manipulation sub-language. In [35], authors propose a query language for video retrieval enabling queries at both image and semantic levels, for retrieving videos with both exact matching and similarity matching.

One of the latest attempts in providing a unified language for MIR is represented by the MPEG Query Format [1]. MPQF is part of the MPEG-7 standard, and provides a standardized interface for MIR systems based on the XML metadata representation formats. MP7QF derives from the well-known set of

XML-based query languages (e.g., *XPath* and *XQuery*), from which it inherits both the syntax and the semantics. MP7QF provides a rich set of multimedia query types (e.g., *QueryByMedia*, *QueryByDescription*, *QueryByFreeText*, *SpatialQuery*, *TemporalQuery*, *QueryByXQuery*, etc.), and specifies a set of precise output parameters describing the response of a multimedia query request by allowing the definition of the content as well as structure of the result set. MP7QF can also be extended with novel query operators. For instance, in [79] authors introduce the *SpatioTemporalOperator*.

## 7 Examples of Research and Commercial MIR Solutions

In this Section we overview a number of research projects that have prototyped the architecture and techniques of a MIR solution, as well as a sample of commercial systems that enable querying multimedia content.

### 7.1 European and Regional Research Projects

**PHAROS** [10] is an Integrated Project of the Sixth Framework Program (FP6) of the European Community. PHAROS has developed an extensible platform for MIR, based on the automatic annotation of multimedia content of different nature: audio, images and video. PHAROS content annotation process has a plug-in architecture: the content process can be defined (with a proprietary tool) and deployed in a distributed manner, possibly incorporating external components, invoked as web services. On top of the PHAROS platform two showcase applications, one for fixed Internet and one for mobile networks, have been prototyped.

**VITALAS** [18] is an FP6 Research Project which has implemented a prototype system for the intelligent access to multimedia professional archives. VITALAS was conceived as a B2B tool to develop and validate technologies applicable to large consumer-facing MIR search engines. The main objective is to enable scalable cross-media indexing and retrieval, as well as methods for content aggregation through the automatic extraction of metadata. VITALAS has produced a prototype implementation of automatic annotation algorithms, visual interfaces for searching in large audio-visual archives, and search personalization techniques.

**THESEUS** [73] is an ongoing German research program aimed at developing a new Internet-based infrastructure to better exploit the knowledge available on the Internet. To this end, application-oriented basic technologies and technical standards are being developed and tested. For instance, the THESEUS project created and supports the Open Source project SMILA [66] (Semantic Information Logistics Architecture), a reliable, standardized industrial strength enterprise framework for building searches solutions to various kinds of information (i.e. accessing unstructured information). Since June 2008, SMILA is an official project of the Eclipse Foundation.

**Quaero** [74] is a French collaborative research and development program that aims at developing multimedia and multilingual indexing, processing, and management tools to build general public search applications on large collections of multimedia information (multilingual audio, video, text, etc.). The challenge of Quaero is to integrate search and indexing components with audio/images/video

processing techniques, semantic annotation methodologies and automatic machine translation technologies, with a specific focus on improving the quality and relevance of these later technologies and techniques.

## 7.2 Examples of Commercial MIR Systems

**Midomi** [47] is an example of audio processing technologies applied to music search engine. The interface allows users to upload voice recordings of public songs, and then to query such music files by humming or whistling. Another similar application is **Shazam** [65]. Shazam is a commercial music search engine that enables users to identify tunes using their mobile phone. The principle consists in using its mobile phone to record a sample of few seconds of a song from any source (even with bad sound quality) and the system returns the identified song with the necessary details: artist, title, album, etc. Similar systems for music search are also provided by **BMat** [8]. **Voxalead**<sup>TM</sup> [77] is an audio search technology demonstrator implemented by Exalead to search in TV news, radio news, and VOD programs by content. The system uses a third-party speech-to-text transcription module transcribe political speeches in several languages.

The field of image search technologies also appears to be mature. **Google Images** [26] and **Microsoft Bing** [48], for instance, now offer a “*show similar images*” functionality, thus proving the scalability of content-based image search on the Web. Other notable examples of image MIR engine are **Tiltomo** [68], which also performs search according to the image theme, and **SAPIR** [33], a search engine developed within the homonymous EU-founded project which also provide geographic and video search. **Blinkx** [7] is another example of search engine on videos and audios streams. Blinkx, like Voxalead<sup>TM</sup>, uses speech recognition to match the text query to the video or audio speech content. Blinkx represents an example of mature video MIR solution as they claim to have over 30 million hours of video indexed.

## 8 Conclusion and Perspectives

In this chapter we presented the problem of Multimedia Information Retrieval, highlighting its challenges, major technical issues, and application areas, and we proposed a reference architecture unifying the aspects of content processing and querying. Then, we provided a survey on the existing research and commercial solution for multimedia search, showing the existence of several mature MIR technologies, products, and services.

In a context where the production of content has become massive thanks to the availability of cheap and high-quality recording devices, MIR solutions represent a fundamental tool for the access to content collections.

MIR systems could benefit from the Search Computing approach in various ways:

- At the architecture level, a MIR system is often implemented on top of a set of distributed Web services, each specialized in a different content analysis and/or query processing technique. In such a distributed scenario, MIR query processing resembles the computation of a multi-domain query: the query “find news clips where President Obama discusses the Health Insurance

Reform could be resolved by joining the results, ranked by confidence, of two metadata sources, one capable of locating the face of President Obama in a video and one able to process the text transcriptions to identify the topic of a discussion.

- At the user interface level, the Liquid Query paradigm could be used to enable the exploration of large multimedia collection. The user could start with a focused query (e.g., a keyword query on the text transcripts of news clips) and then expand the query by joining other metadata sources, exposed as service interfaces: e.g., looking for other videos featuring the same speaker, or produced by the same media agency.

On the other hand, MIR can also extend the capabilities of Search Computing systems by enabling new ways of matching and ranking in multi-domain queries. For instance:

- **MIR Systems as Domain-Specific Search Engines:** MIR systems can be adopted in Search Computing as a special category of ranked search services: for instance, in a multi-domain query for a market analysis application, a text transcription search engine could be wrapped as a service interface for selecting and ranking news clips according to the probability that they deal with a given company, provided in input as a keyword.
- **MIR Operations as Query Operators:** including non-textual content items as query conditions can enrich the expressive power of Search Computing systems. For instance, users can express their information need as images or audio files, leaving to the analysis and annotation operations the task of extracting, in a textual form, the concept to use as a join condition (e.g., the name of a person given an image of its face). Even more interestingly, a multi-domain query could directly exploit content similarity to compute joins: in a trip planning multi-domain query, the destinations could be joined to the result of the query based on their similarity to the user's favourite beach, supplied in input as an image.

## References

- [1] Adistambha, K., Döller, M., Tous, R., Gruhne, M., Sano, M., Tsinaraki, C., Christodoulakis, S., Yoon, K., Ritz, C., Burnett, I.: The MPEG-7 Query Format: A New Standard in Progress for Multimedia Query by Content. In: Proceedings of the 7th International IEEE Symposium on Communications and Information Technologies (ISCIT 2007), pp. 479–484 (2007)
- [2] Adobe Premiere (2009), <http://www.adobe.com/products/premiere/>
- [3] Baeza-Yates, R., Ribeiro-Neto, B.: Modern Information Retrieval, 1st edn. Addison Wesley, Reading (1999)
- [4] Baldi, A., Murace, R., Dragonetti, E., Manganaro, M., Guerra, O., Bizzi, S., Galli, L.: Definition of an automated Content-Based Image Retrieval (CBIR) system for the comparison of dermoscopic images of pigmented skin lesions. Biomed. Eng. Online (2009)
- [5] Barbieri, M., Agnihotri, L., Dimitrova, N.: Internet Multimedia Management Systems IV. In: Proceedings of the SPIE, vol. 5242, pp. 1–13 (2003)



- [6] Beitzel, S.M., Jensen, E.C., Grossman, D.A.: Retrieving OCR Text: A Survey of Current Approaches. In: Symposium on Document Image Understanding Technologies, SDUIT (2003)
- [7] Blinkx – Video Search Engine (2009), <http://www.blinkx.com/>
- [8] BMat - 2009 (2009), <http://www.bmat.com/>
- [9] Bozzon, A., Brambilla, M., Fraternali, P.: Model-Driven Design of Audiovisual Indexing Processes for Search-Based Applications. In: 7th IEEE International Workshop on Content-Based Multimedia Indexing, pp. 120–125. IEEE Press, New York (2009)
- [10] Bozzon, A., Brambilla, M., Fraternali, P., Nucci, F., Debal, S., Moore, E., Neidl, W., Plu, M., Aichroth, P., Pihlajamaa, O., Laurier, C., Zagorac, S., Backfried, G., Weinland, D., Croce, V.: Pharos: an audiovisual search platform. In: Proceedings of the 32nd international ACM SIGIR Conference on Research and Development in information Retrieval, SIGIR 2009, Boston, MA, USA, July 19 - 23, p. 841. ACM, New York (2009)
- [11] Buchenwald demonstrator, University of Twente (2009), <http://vuurvink.ewi.utwente.nl:8080/Buchenwald/>
- [12] Caringella, N., Zoia, G., Mlynek, D.: Automatic genre classification of music content: a survey. IEEE Signal Processing Magazine 23(2), 133–141 (2006)
- [13] Carrato, K.S.: Temporal video segmentation: a survey. Signal Processing: Image Communication 16, 477–500 (2001)
- [14] Cees, G.M.: Concept-Based Video Retrieval. Foundations and Trends in Information Retrieval 4(2), 215–322 (2009)
- [15] Cotsaces, C., Nikolaidis, N., Pitas, I.: Video Shot Boundary Detection and Condensed Representation: A Review. IEEE Signal Processing Magazine (2006)
- [16] Delve Networks - Online Video Platform and Content Management (2009), <http://www.delvenetworks.com/>
- [17] Devlin, B., Wilkinson, J.: The Material Exchange Format. In: Gilmer, B. (Hrsg.) File Interchange Handbook, pp. 123–176. Elsevier Inc., Focal Press (2004)
- [18] Diou, C., Papachristou, C., Panagiotopoulos, P., Stephanopoulos, G., Dimitriou, N., Delopoulos, A., Rode, H., Aly, R., de Vries, A.P., Tsirikas, T.: VITALAS at TRECVID 2008. In: Proceedings of the 6th TREC Video Retrieval Evaluation Workshop, Gaithersburg, USA, November 17-18 (2008)
- [19] Dublin Core Metadata Initiative (2009), <http://dublincore.org/>
- [20] Empora Online Shop (2009), <http://www.empora.com>
- [21] Eu, H., Hedge, A.: Survey of continuous speech recognition software usability. Cornell University, Ithaca, NY (1999), <http://ergo.human.cornell.edu/AHProjects/Hsin99/Voice%20Recognition%Paper.pdf> (retrieved April 5, 2004)
- [22] Eyealike platform for facial similarity (2009), <http://www.eyéalike.com/home>
- [23] Facesaerch (2009), <http://www.facesaerch.com/>
- [24] Geurts, J., van Ossenbruggen, J., Hardman, L.: Requirements for practical multimedia annotation. In: Workshop on Multimedia and the Semantic Web Heraklion, Crete, pp. 4–11 (2005)
- [25] Google Election Video Search (2009), <http://googleblog.blogspot.com/2008/07/in-their-own-words-political-videos.html>
- [26] Google Images (2009), <http://images.google.com>
- [27] Google Picasa (2009), <http://picasa.google.com/>
- [28] Hanbury, A.: A survey of methods for image annotation. Journal of Visual Languages and Computing 19(5), 617–627 (2008)

- [29] Henrich, A., Robbert, G.: Combining multimedia retrieval and text retrieval to search structured documents in digital libraries. In: Proc. 1st DELOS Workshop on Information Seeking, Searching and Querying in Digital Libraries, Zurich, Switzerland, vol. 01/W001 (2000)
- [30] Henrich, A., Robbert, G.: POQLMM: A Query Language for Structured Multimedia Documents. In: Proceedings of the First International Workshop on Multimedia Data and Document Engineering, Lyon, France, pp. 17–26 (2001)
- [31] Japan Electronics and Information Technology Industries Association: Exchangeable image file format for digital still cameras: EXIF. Version 2.2 (2002)
- [32] ID3 (2009), <http://www.id3.org/>
- [33] IST SAPIR Large Scale Multimedia Search and P2P (2009), <http://safir.isti.cnr.it/index>
- [34] International Press Telecommunications Council (2009), <http://www.iptc.org/IPTC4XMP/>
- [35] Le, T.H., Thonnat, M., Boucher, A., Bremond, F.: A Query Language Combining Object Features and Semantic Events for Surveillance Video Retrieval. In: Proceedings of Advances in Multimedia Modeling, 14th MMM Conference, Kyoto, Japan, pp. 307–317 (2008)
- [36] Learning Object Metadata (2009), <http://ltsc.ieee.org/wg12/>
- [37] Lew, M., et al.: Content-Based Multimedia Information Retrieval: State of the Art and Challenges. *ACM Transactions on Multimedia Computing, Communications, and Applications* 2(1) (2006)
- [38] Liu, Y., Zhang, D., Lu, G., Ma, W.: A survey of content-based image retrieval with high-level semantics. *Pattern Recogn.* 40(1), 262–282 (2007)
- [39] LSCOM Lexicon Definitions and Annotations (2009), <http://www.ee.columbia.edu/ln/dvmm/lscom/>
- [40] LTU technologies (2009), <http://www.ltutech.com/>
- [41] Martínez, J.M.: MPEG-7 Overview (version 10), ISO/IEC JTC1/SC29/WG11N6828, Palma de Mallorca (2004)
- [42] Manjunath, B.S., Salembier, P., Sikora, T.: Introduction to MPEG-7: Multimedia Content Description Interface, 396 p. Wiley, Chichester (2002)
- [43] Maragos, P., Potamianos, A., Gros, P.: Multimodal Processing and Interaction, Audio, Video, Text. *Multimedia Systems and Applications*, vol. 33. Springer, Heidelberg (2008)
- [44] Marsden, A., Mackenzie, A., Lindsay, A.: Tools for Searching, Annotation and Analysis of Speech, Music, Film and Video; A Survey. *Literary and Linguistic Computing* 22(4), 469–488 (2007)
- [45] Media RSS (2009), [http://en.wikipedia.org/wiki/Media\\_RSS](http://en.wikipedia.org/wiki/Media_RSS)
- [46] Meyers, O.C.: A Mood-Based Music Classification and Exploration System, MS Thesis, Massachusetts Institute of Technology (MIT), USA (2007)
- [47] MiDoMi (2009), <http://www.midomi.com/>
- [48] Microsoft Bing (2009), <http://www.bing.com/images>
- [49] MPEG Industry Forum (2009), <http://www.m4if.org/>
- [50] Ngo, C., Chan, C.: Video text detection and segmentation for optical character recognition. *Multimedia Systems* 10(3), 261–272 (2004)
- [51] Petrovska-Delacrétaz, D., El Hannani, A., Chollet, G.: Text-Independent Speaker Verification: State of the Art and Challenges. In: Stylianou, Y., Faundez-Zanuy, M., Esposito, A. (eds.) COST 277. LNCS, vol. 4391, pp. 135–169. Springer, Heidelberg (2007)
- [52] Pictron Solutions (2009), <http://www.pictron.com/>

- [53] Pixta, Visual search technologies (2009), <http://www.pixsta.com/>
- [54] Pluggd Podcast Search Engine (2009), <http://www.pluggd.tv/>
- [55] Podcast (2009), <http://en.wikipedia.org/wiki/Podcasting>
- [56] Podscope Podcast Search Engine (2009), <http://www.podscope.com/>
- [57] Potamitis, I., Ganchev, T.: Generalized recognition of sound events: Approaches and applications. *Studies in Computational Intelligence*, vol. 120, pp. 41–79. Springer, Heidelberg (2008)
- [58] Radio Oranje speech search, Univeristy of Twente (2009), <http://hmi.ewi.utwente.nl/choral/radiooranje.html>
- [59] Radke, R.J., Andra, S., Al-Kofahi, O., Roysam, B.: Image change detection algorithms: a systematic survey. *IEEE Transactions on Image Processing* 14(3), 294 (2005)
- [60] Rasheed, Z., Shah, M.: Scene detection in Hollywood movies and TV shows. In: *Proceedings of the IEEE Computer Vision and Pattern Recognition Conference* (2003)
- [61] Recordare (2009), <http://www.recordare.com/xml.html>
- [62] RSS (2009), [http://en.wikipedia.org/wiki/RSS\\_file\\_format](http://en.wikipedia.org/wiki/RSS_file_format)
- [63] Sacco, S.M., Tzitzikas, Y.: *Dynamic Taxonomies and Faceted Search, Theory, Practice, and Experience*. The Information Retrieval Series, vol. 25, p. 340. Springer, Heidelberg (2009)
- [64] Salton, G., Wong, A., Yang, C.S.: A vector space model for automatic indexing. *Commun. ACM* 18(11), 613–620 (November 1975) (2003)
- [65] SHAZAM (2009), <http://www.shazam.com/>
- [66] SMILA (2009), <http://www.eclipse.org/smila/>
- [67] SMEF (2009), <http://www.bbc.co.uk/guidelines/smef/>
- [68] TILTOMO (2009), <http://www.tiltomo.com/>
- [69] Tineye, Image Search Engine (2009), <http://tineye.com/>
- [70] The 3GP video standard (2009), <http://www.3gp.com/>
- [71] The DAML Ontology Library (2009), <http://www.daml.org/ontologies>
- [72] The Internet Movie Database (2009), <http://www.imdb.com>
- [73] The Theseus programme (2009), <http://theseus-programm.de>
- [74] The Quaero Program (2009), <http://www.quaero.org>
- [75] Turaga, P., Chellappa, R., Subrahmanian, V.S., Udrea, O.: Machine recognition of human activities: A survey. *IEEE Transactions on Circuits and Systems for Video Technology* 18(11), 1473–1488 (2008)
- [76] Typke, R., Wiering, F., Veltkamp, R.C.: A survey of music information retrieval systems. In: *ISMIR 2005*, pp. 153–160 (2005)
- [77] Voxlead<sup>TM</sup> (2009), <http://voxlead.labs.exalead.com>
- [78] Wang, C.C., Wang, J., Li, J., Sun, J.G., Shi, S.: MuSQL: A Music Structured Query Language. In: Cham, T.-J., Cai, J., Dorai, C., Rajan, D., Chua, T.-S., Chia, L.-T. (eds.) *MMM 2007*. LNCS, vol. 4352, pp. 216–225. Springer, Heidelberg (2006)
- [79] Wattamwar, S.S., Ghosh, H.: Spatio-temporal query for multimedia databases. In: *Proceeding of the 2nd ACM Workshop on Multimedia Semantics (MS 2008)*, pp. 48–55. ACM, New York (2008)
- [80] Yilmaz, A., Javed, O., Shah, M.: Object tracking: A survey. *ACM Comput. Survey* 38(4) (2006)
- [81] Yu, G., Chen, Y., Shih, K.: A Content-Based Image Retrieval System for Outdoor Ecology Learning: A Firefly Watching System. In: *International Conference on Advanced Information Networking and Applications*, vol. 2, p. 112 (2004)
- [82] Zhao, W., Chellappa, R., Phillips, P.J., Rosenfeld, A.: Face recognition: A literature survey. *ACM Comput. Surv.* 35(4), 399–458 (2003)

## **Part III**

# **Issues in Search Computing**

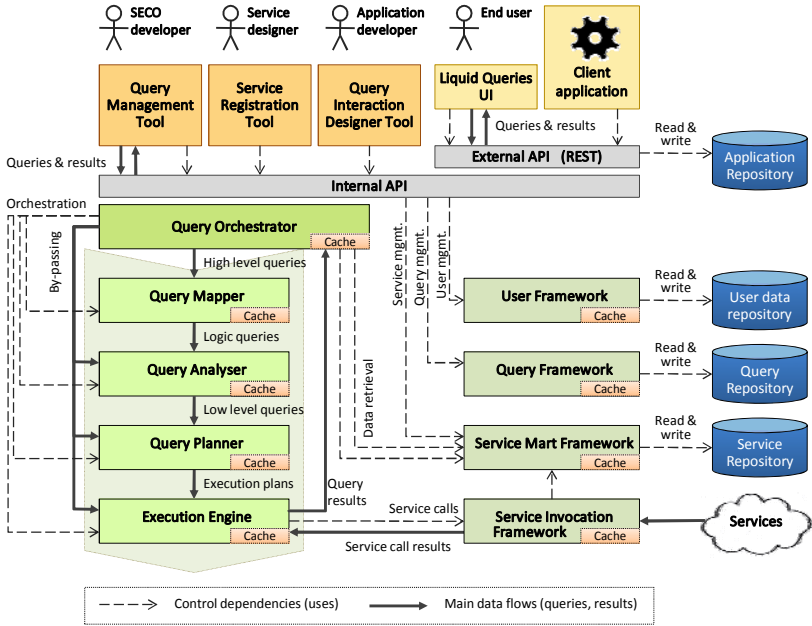
## Introduction to Part III: Search Computing in a Nutshell

Prior to delving into chapters discussing search computing in greater detail, we give a bird's eye view of its various phases and components, by providing an architectural view of the search computing prototyping environment.

Search computing systems support their users in making multi-domain queries; for instance, “Where can I attend a DB scientific conference close to a beautiful beach reachable with cheap flights?”. A system *decomposes* the query into sub-queries (in this case: “Where can I attend a DB scientific conference?”; “which place is close to a beautiful beach?”; “which place is reachable from my home location with cheap flights?”) and maps each sub-query to a domain-expert server (in this case, calls to servers named “Conference”, “Tourism”, “Low-Cost-Flights”); it then *analyzes* the query and translates it into an internal format, which is then optimized, thereby yielding to an optimal *plan* for query execution; plan execution is supported by an *execution engine*, which submits service calls to services through a *service invocation framework*, builds the *query results* by combining the outputs produced by service calls, computes the *global rankings* of query results, and outputs query results in an order that reflects, although with some approximation, their global ranking.

These transformation steps are shown in the bottom-left side of Fig. 1; they are performed by the *query mapper*, *query analyzer*, *query planner*, and *execution engine*, under the responsibility of a query *orchestrator* that starts query execution and collects query results. The figure shows that each of the four modules directly accepts user-provided input through suitable interfaces; in this way, prototype implementation in search computing can take place bottom-up, by starting with the *execution engine*, which can execute a given plan, then adding the *query planner*, which produces the optimal plan for a given internal query, then adding the *query analyzer*, which reads an abstract query, checks that the query is legal, and produces an internal query; and finally adding a *query mapper*, capable of decomposing a multi-domain query into several domain-specific queries. In this book we do not address query mapping, while we address the other steps. The search computing prototyping architecture is currently well-defined in terms of interactions and of functionalities; prototypes will be delivered throughout the course of the SeCo Project.

Services are made available to search computing through a standard format, called *service mart*; by this term we mean an abstraction that masks the different implementation styles of services and is tailored to the specific need of exposing *search services* – i.e., services whose primary purpose is to produce ranked lists of results. Moreover, service marts offer a classification of service properties (that represent either the call or the result of a service invocation; output results may represent the ranking values) and a definition of composition patterns allowing the combination of service marts.



**Fig. 1.** Overview of the Search Computing framework

Search computing users mainly belong to two categories. *End users* can only launch predefined applications and submit input to them through forms; *expert users* may also compose queries in the context of repositories of service marts and of their composition patterns (we say in such cases that users can build liquid queries, where their liquid nature comes from the fact that queries extend upon service marts more or less as stains over surfaces). In both cases, however, we expect users to have some experience in data analysis (similar, e.g., to the basic skills required by spreadsheets) and we expect them to use such skills in manipulating results, which are shown in tabular format, and can be dynamically augmented online – we call them liquid results to highlight the dynamic and plastic nature of such results, which can be manipulated by means of user controls.

Tools are intended to support three kinds of experts:

- *Service designers* register data sources in the system through the Service Mart Framework, by either interacting with existing Web services, or by exposing existing data sources, or by wrapping existing Web pages. They play the role of “data providers”.
- *Application developers* preselect some of the services and configure them so as to turn them into applications; specifically, they build user interfaces which either expose service marts and their connections to expert users or simple forms accepting typed input to end users. They play the role of “data brokers”.
- *SeCo developers* install, open and configure the SeCo modules on suitable hardware resources and may perform fine tuning (or creation from scratch) of query and execution plans.

The upper part of Figure 1 shows that the tools provided to the designer and developer communities plug to an internal API, while end-user applications and interfaces in turn are accessible via an external API and therefore callable from any client environment. Finally, the right part of Figure 1 shows three kinds of repositories, called *service repositories* (i.e., registries of search services registered in the framework), *query repositories* (i.e., queries that have been saved by the user for subsequent restore operations), and *user data repositories* (i.e., profiles and administrative information). An additional *application repository* loads applications and stores the user's interactions, so as to be able to remember and re-apply such interactions to new queries or to new results.

The various architectural elements forming the search computing prototype architecture defined above are described in different, autonomous chapters.

Chapter 9 deals with **service marts**, a novel concept for enabling the engineering and deployment of search services, i.e., of services whose main feature is the ability to respond to ranked results organized by chunks (so as to enable a fine-grain control by the execution engine). Such results are produced by interacting with concrete data sources, which are made available through service interfaces, wrappers, or direct access to extensional data collections (databases, excel files, and so on). Thus, service marts are conceptual abstractions providing information hiding, mapped to service interfaces which directly interact with concrete data sources.

Chapter 10 describes our **framework for query execution**; specifically it addresses the description of a query language for search computing, then the mapping of queries to service interfaces, then the composition methods that have been defined so far in search computing under the classical format of join methods, suitably extended to the search and web context. This chapter discusses query formulation and optimization up to the choice of join methods.

Chapter 11 deals with **ranking aggregation** in its most general formalization, and shows how ranking aggregation methods can be adapted to search computing in generating a join method which is capable of guaranteeing that the top-k results are selected.

Chapter 12 describes a **flexible architecture for search computing** (named *Panta Rhei*) which includes suitable abstractions for data production, consumption, and caching, with both data-driven and event-driven synchronization. Operations and flows of the *Panta Rhei* model are described at a high level, but they are designed for supporting the scalable execution of search computing queries in a variety of deployment architectures.

Chapter 13 shows a paradigm for making search computing queries, called **liquid queries**, that can be articulated upon a flexible architecture, where a liquid query is capable of run-time modification by the addition or dropping of sub-queries and by drilling down and rolling up information, much in the same way as with a data cube expressing the results in data analysis environments.

Chapter 14 shows how to **build and deploy applications** by means of a software engineering environment involving both "data providers", who will register service marts, and "data brokers", who will assemble applications. SeCo servers can be deployed upon a variety of computing architectures, hinting to future prototypes running upon highly scalable architectures and/or cloud computing systems. The

chapter also discusses the business models that may favor the spreading of both data providers and data brokers.

Chapter 15 discusses **ranking opportunities in the context of life sciences**, which are characterized by a wide use of ranked information, thereby anticipating some of the specific issues featured by an appealing search computing application.



# Chapter 9: Service Marts

Alessandro Campi<sup>1</sup>, Stefano Ceri<sup>1</sup>, Georg Gottlob<sup>2</sup>, Andrea Maesani<sup>1</sup>,  
and Stefania Ronchi<sup>1</sup>

<sup>1</sup> Politecnico di Milano, Italy

{campi,ceri,maesani,ronchi}@elet.polimi.it

<sup>2</sup> University of Oxford, United Kingdom

georg.gottlob@comlab.ox.ac.uk

**Abstract.** The use of patterns in data management is not new: in data warehousing, data marts are simple conceptual schemas with exactly one core entity, describing facts, surrounded by multiple entities, describing data analysis dimensions; data marts support special analysis operations, such as roll up, drill down, and cube. Similarly, service marts are simple schemas which match "Web objects" by hiding the underlying data source structures and presenting a simple interface, consisting of input, output, and rank attributes; attributes may have multiple values and be clustered within repeating group. Service marts support Search Computing operations, such as ranked access and joins. When objects are accessed through service marts, responses are ranked lists of objects, which are presented subdivided in chunks, so as to avoid receiving too many irrelevant objects – cutting results and showing only the best ones is typical of search services. This chapter includes a survey of service definition standards (discussing the standards for service description and the current state-of-the-art for service registration and discovery), then introduces a formal definition of service marts and of connection patterns at the conceptual, logical, and physical levels. Then, we show how service marts can be implemented, by taking into account different kinds of data sources, and taking advantage of components (written in Java and SQL) and tools (such as a materialize specifically developed to help service mart implementation). We use such components and tools to build a collection of services used in a running example throughout the chapters of this part.

## 1 Introduction

The goal of Search Computing is to support search service integration, but a prerequisite for setting such goal is the availability of a large number of valuable search services. With a passive attitude, we could just wait for SOA (Service Oriented Architecture) to become widespread, and then use available services within our framework. However, few software services are currently designed to support search, and moreover a huge number of valuable data sources (the so called "long tail" of Web information) are not provided with a service interface. In this chapter, we therefore focus on the important issue of publishing service interfaces suitable for

Search Computing on the data sources, so as to facilitate their use on the Web and at the same time to create the premises for their integration within the Search Computing framework.

At the basis of our work, we observe the pervasive role of software services and SOA; a convincing argument is the prominent role given to Software and Services in Call 5 of the EU-funded Seventh Framework Programme (FP7), whose goal is to achieve an “Internet of the Future, where organizations and individuals can find software as services on the Internet, combine them, and easily adapt them to their specific context [22].” While the SOA principles are becoming widespread, however, we observe several ways in which principles are turned into standards, languages, and programming styles. Thus, there is room to orient the SOA vision in a variety of directions, and this paper emphasizes the relevance of supporting ordered queries upon data sources. Such emphasis is new and very relevant to our Search Computing goal.

The key success factor in the building of an abstract model useful to describe online data sources is the recognition of a simple pattern, supporting a data publication strategy, which should be “just enough” expressive to support data source publishing, i.e., neither too simple, nor too complex. The search of patterns for easing data publication for specific contexts is not new; the most well known data publication pattern is the so called “data mart”, used in data warehousing as conceptual schemas for driving data analysis. Data marts [6] are simple schemas having one core entity, describing facts, surrounded by multiple entities, describing the dimensions of data analysis. Such subschema allows a number of interesting operations for data selection and aggregation (e.g. data cubes, rollup, drilldown) whose semantic definition is much simplified by data characterization as either fact or dimension and by the regular structure of the schema.

Analogously, a “Web mart” [7] is a pattern introduced in the Web design community to characterize the role played by data items in data-intensive Web applications. Web marts have a central entity, the core concept, which describes a collection of core objects, surrounded by other entities which are classified as “access entities”, enabling selection of core objects through navigation, and “detail entities”, describing core objects in greater detail. Thus it is possible to drive a design process that produces first-cut standard Web applications (e.g. sales, inventories, travels, and so on).

Following an analogy with these two cases, in the framework of Search Computing we define a **service mart** as the data abstraction for data source publication and composition. The goal of a service mart is to ease the publication of a special class of software services, called search services, whose responses are ranked lists of objects; moreover, pairs of service marts are augmented with “connection” patterns, so as to support queries over several service marts. Every service mart is mapped to one “Web object” available on Internet; therefore, we may have service marts for “hotels”, “flights”, “doctors”, and so on. Thus, service marts are consistent with a view of the “internet of objects” which is gaining popularity as a new way to reinterpret concept organization in the Web and go beyond the unstructured organization of Web pages.

Service marts support queries, therefore the properties (or attributes) of a service mart logically belong to two types: those accepting “input” values from queries, and those providing “output” values (i.e. results) to queries. The distinction is not crisp,

because a given service mart can offer several implementations and therefore an input attribute in one implementation can be in output in another. However, if we consider a specific service implementation, then the distinction is clear. Such distinction is useful for characterizing the role of attributes within queries, hence for composing queries which consist of multiple service marts. Moreover, the most interesting service marts are those offering ranked output. Such ranking is either opaque (being known to the search service but not published in its standard interfaces), or explicitly associated with given output attributes (either one or more; in such cases, ranking is typically a linear combination of the values exposed by given output attributes). When ranking is explicit, some of the output attributes of a service mart contain data which allow sorting the results of service calls.

Service marts are abstractions; publishing a service mart entails bridging an abstract description to several concrete implementations of services. Indeed, implementing the service mart “hotel” may require the mapping to several data sources, each one configured either as Web service (and accessible through APIs with somewhat different features) or as a materialized data collection (e.g., data provided by hotel portals and wrapped as a table by using suitable tools). Thus, the service mart concept offers an abstraction for giving a “regular” view of the world, together with a method and associated technology for building such a regular view out of concrete data sources.

In the rest of this chapter, we survey the state-of-the-art of SOA deployment, with a specific emphasis on data-intensive (as opposed to process-intensive) uses of SOA; then Section 3 defines the notion of service marts (at a conceptual, logical, and physical level) and of composition pattern (also at three levels). Section 4 illustrates the registration process of service marts and applies such process to service marts of a running example that will be next used thorough the various chapters; finally, Section 5 illustrates best practices of service mart development in the context of Web services, of wrapped data sources, and of materialized data sources.

## 2 Service-Oriented Architectures for Data Publishing

### 2.1 Service Description

The software service paradigm addresses the issue of making software systems more easily composable and interoperating, hence many successful standards for services have been progressively developed, either *de jure* (within standardization bodies, such as W3C) or *de facto* (promoted by companies). In this section we briefly name the most popular ones, referring the interested reader to courses and books (e.g. [2]).

Services are the result of a long research stream in software development, rooted in previous attempts to support interoperability, such as CORBA (Common Object Request Broker Architecture), DCOM (Distributed Component Object Model) or RMI (Java/Remote Method Invocation). These standards were associated with competing computational models for distributed components.

In the end, the most relevant standards are rather agnostic about the underlying computational model, and describe instead other aspects of Web services, such as their interfaces or message formats. The most popular interface definition standard is

WSDL (Web Service Description Language) [32], a machine-processable format describing the public interface offered by the Web service (with emphasis on its operations and their parameters). The most popular message formats for exchanging information to and from Web services follow the SOAP (initially: Simple Object Access Protocol) standard [27], based on HTTP and XML. These standards are very verbose. For this reason a lot of work has been done in the automatic generation of code that manages web Services. The most interesting frameworks are Spring [28], Apache Axis [20] and CXF [21]. SOAP services are also enriched by additional functionalities (e.g. security, transaction, ...) thus making SOAP services a mature platform especially for enterprise applications.

More recently, REpresentational State Transfer (RESTful) Web services [25] have been regaining popularity, particularly for use on Internet. By using the PUT, GET and DELETE HTTP methods, alongside POST, these are often better integrated with HTTP and web browsers than SOAP-based services. They do not require XML messages or WSDL definitions.

Services act as components which can be put together to form software applications according to given specifications (or service choreographies and orchestrations, see chapter 12 and [2]); a software application, at a given time, is built by a particular service composition, which should be consistent with such specification. When running composite services on the Web, however, each sub service is autonomous; if we consider most services which are provided on the Web, the provider may remove, change or update their services without giving notice, and this may cause faults during application execution. Therefore, an important aspect of current research in web services is to guarantee properties of service compositions in dynamic contexts where individual services can change or fail, but at the same time the application can react by resorting to alternative compositions and possibly alternative services. Exception handling and dynamic composition in the context of web services is still an open research issue [15].

Another important research area is concerned with the exploitation of semantic Web services [12], i.e. the ability to build applications starting from abstract descriptions of the user's goals and then building legal service compositions by means of reasoning technologies. Standards such as WSDL or REST enable the invocation of services but do not describe their semantics; therefore, the Semantic Web has extended description standards so as to add semantics. One such extension is provided by SAWSDL (Semantic Annotations for WSDL and XML Schema) [26], a mechanisms for adding semantic annotations to WSDL components. SAWSDL does not specify a language for representing the semantic models, e.g. ontologies, but it only requires that the semantic concepts be identifiable via URI references, and provides mechanisms by which concepts from the semantic models that are defined either within or outside the WSDL document can be referenced from within WSDL components as annotations. A different semantic extensions to web service descriptions is WSFL [16], while DAML-S [3] applies reasoning technologies of the semantic web to service description.

The Web Service Modeling Framework (WSMF) [11] is a modeling framework centered around two complementary principles: decoupling and mediation, which respectively support separation and interaction. WSMO [9] has four main components: ontologies (providing the terminology used by elements to describe the relevant

aspects of the domains), Web services (the computational entities able to provide access to services), goals (user desires w.r.t. the requested functionality), and mediators (that describe elements that handle interoperability problems between different WSMO elements). Mediators act as the core concept to resolve incompatibilities on the data, process and protocol level, i.e., in order to resolve mismatches between different used terminologies (data level), in how to communicate between Web services (protocol level) and on the level of combining Web services (process level). WSMO uses core concepts of the Web, such as using URI (Universal Resource Identifier) for unique identification of resources and Namespaces for denoting consistent information spaces, supports XML and other W3C Web technology recommendations, as well as the decentralization of resources.

The positioning of Search Computing relative to the Semantic Web deserves a discussion. The Semantic Web vision is that of machines that substitute to humans in the whole chain going from the high-level expression of goals down to the fulfilment of the goals; it entails steps of discovery, matching, mediation, negotiation, and delivery of information. Search computing aims at solving a simpler problem, in which search sources are pre-selected, the most appropriate patterns are defined for their composition, and queries are initially confined to use such patterns. Moreover, search computing has a well-defined interaction context: user interfaces support interactions whose purpose is to make ranking criteria explicit in the presence of ambiguity. Thus, Search Computing is focusing on a much simpler problem than the Semantic Web in its overall approach. However, our future work aims at leveraging query interfaces and making them more and more close to natural language, and in such cases semantic service annotations will become essential. For easing the transition to this future goal, we already support optional tagging and annotations in the service mart registration environment, and we may be able to take advantage of the Linking Open Data initiative (<http://linkeddata.org/>), whose aim is to make information on the Web available to machines (through RDF and URLs). This information can potentially represent an additional source of data for Search Computing (many useful sources are already available as linked data, e.g. dbpedia, geonames, pubmed, citeseer).

## 2.2 Service Registration

At the heart of service-oriented computing is the possibility to cluster information about services within so-called “service registries” that list (and sometimes connect and mediate) service providers. Web service registries allow clients or applications to access a wide range of Web services that match specific search criteria.

UDDI (Universal Description Discovery and Integration) [30] is a standard for allowing service providers to share information stored within a registry and then to discover the most appropriate service based upon its description, which we consider in this section by focusing on registration aspects only. A UDDI registry offers services for managing information describing service providers, service implementations, and service metadata. Three types of information are included within UDDI registries.

- White pages describe basic contact information and identifiers such as organization name, address, contact information, and other unique identifiers.
- Yellow pages describe Web Services in terms of categorizations allowing the classification of a Web Service based upon its category.
- Finally, green pages describe the service in terms of technological information, such as its behaviours and operations.

UDDI specifications include a SOAP APIs, for supporting the querying and publishing of information, an XML representation for the registry data model and the SOAP message formats, and WSDL interface definitions of the services for interacting with the UDDI registry.

In recent years, several Web service portals or directories have emerged:

- XMethods [34] is probably the oldest reference for looking to publicly available Web Services. It only provides a simple (long) list of Web Services and a details page with some basic information for every service.
- RemoteMethod Web Services directories [24] support finding and comparing Web Services from various providers. Services need to be registered by a rather lengthy procedure using a sequence of five Web forms and give detailed information, like the infrastructure where the Web Service is hosted and uptime guarantees. Catalog management follows a business model where service providers can buy banners or pay in order to increase their position within the listings.
- StrikeIron [29] is a marketplace of commercial Web Services and allows browsing by category as well as keyword search. Its mission is to support commercialization of Web Services.
- Woogle [31] searches for UDDI nodes in order to extract web services from them, and focuses on extracting the semantic meaning of web services based on WSDL descriptions, and then presenting the users a search interface that exposes as well the semantic relationships between web services.
- Wsoogle [33] is based on a categorization built on top of the Woogle technology. Similarity evaluation for operations uses operation names and names and types of their parameters.

URBE [18] proposes an algorithm able to evaluate the similarity degree between two Web services (or a query and a Web service) by comparing the related WSDL descriptions. The approach takes into account the relationships between the main elements composing a WSDL description (i.e., portType, operation, message, and part) and, if available, the annotations included in a SAWSDL (Semantic Annotated WSDL) file. The URBE approach has been implemented in a prototype that extends a UDDI registry.

While these examples are useful first sources for looking for services, many service directories that can be found on the Web are not stable, fail to adhere to UDDI and, with time, become unreliable. Moreover, centralized registries suffer from problems associated with having centralized systems such as a single point of failure, and bottlenecks.

## 2.3 Service Discovery

Discovery is [8] “the act of locating a machine-processable description of a Web service that may have been previously unknown and that meets certain functional criteria”. Web service discovery mechanisms allow accessing to service repositories and/or “crawling the Web” in the search for services. Since Web services can be tagged with a wealth of information, methods to narrow the discovery can be quite complicated and use such semantic information.

The main obstacle that WS discovery mechanisms face is the heterogeneity between services. A high level approach is considered by the emerging Web service architecture project of W3C [4]. Different kinds of heterogeneity are classified [13,17], such as: technological heterogeneity (different platforms or different data formats), ontological heterogeneity (difference in terms and concepts describing services), pragmatic heterogeneity (different underlying processes and different support to domain-specific tasks).

Search engines such as Google and Yahoo have become a new source for finding Web services. However, search engines do not easily separate and expose to users the basic service properties (i.e. binding information, operations, ports, service endpoints, among others), as they are instrumented or crawling and indexing generic content. In addition, search engines generally crawl Web pages from accessible Web sites while publicly accessible WSDL documents reside on Web servers; hence they are not designed to be fetched and analyzed by normal crawlers. The work in [1] presents a Web Service Crawler Engine (WSCE), i.e. a crawler whose primary purpose is to seek within sources to collect business and Web service information.

Another recent service discovery technique is based on the Conceptual Situation Spaces (CSS) [10], aiming at the discovery of appropriate semantic Web services representations for a given situational context. In order to achieve such goal, the methods first represent the specific situation characteristics as vectors within geometrical vector spaces, according with the idea of Conceptual Spaces, then they evaluate the semantic similarity between different situations as the Euclidean distance of the corresponding vectors within the space. In this way they allow a similarity-based matchmaking.

## 3 Service Marts

This section gives a top-down view of the definition of service marts, from the conceptual level through the logical to the physical level. It then describes composition patterns (at the conceptual and logical level) and service registration.

### 3.1 Conceptual Level

In a given Search Computing context, every class of Web objects must be represented by a single **service mart**. A service mart is an abstraction that induces a normalization of the attributes describing a class of Web objects. Thus, every service mart definition includes a name and a collection of exposed attributes. Service marts have atomic attributes and repeating groups consisting of a non-empty set of sub-attributes that

collectively define a property of the service mart. Atomic attributes are single-valued, while repeating groups are multi-valued.

Thus, a “Concert” service mart may be defined with five single-value attributes (“Number”, “Theatre”, “City”, “Director”, “Orchestra”), then three multi-valued repeating groups (“Show”, with “Date” and “Time” sub-attributes; “Program”, with “Title” and “Composer” sub-attributes; and “Price List”, with “SeatType” and “Price” sub-attributes). The schema of a repeating group is introduced by one level of parentheses; therefore the above example can be summarized by the schema:

**Concert(Number, Theatre, City, Director, Orchestra, Show(Date, Time),  
Program(Title, Composer), Price-List(SeatType, Price))**

As another example, a “Movie” service mart has five single-value attributes (“Title”, “Director”, “Score”, “Year”, and “Language”) and then three repeating groups “Genres”, “Openings” and “Actor”, each with sub-attributes describing them. The schema is then:

**Movie(Title, Director, Score, Year, Language, Genres(Genre),  
Openings(Country, Date), Actors(Name))**

Attributes and sub-attributes are typed and can be tagged when they are defined. Of course, such a pattern introduces a limitation upon the possible ways of describing reality, which seems rather coercive if one considers the richness of data modeling choices offered by top-down design. But in our framework we do not use a top-down process; rather, we model data sources as they exist, bottom-up, and then we look for their integration; moreover, most data sources have a simple schema, which can be well represented by a one-level nesting. Therefore, the expressive power of service marts seems to be appropriate for its purpose.

Every object instance should be “unique” within its class of reference (since there is a single instance of each “real object” in the world). Such uniqueness can be imposed explicitly when the service mart attributes derive their values from extensional data and one of the properties of such extension is defined as (primary) key; but we cannot assume in general that a key is explicitly available or known. Therefore, we do not define the identification of object instance within the conceptual model, and instead add a system-defined key in the physical model, within a normalized schema associated to each service mart.

The motivation for repeating groups within service marts descends from the following goal: we want to associate service marts with properties which can be composite but not too complex. A repeating group serves the purpose of embedding many-valued properties (such as the “actors” of a “movie”) within the object instances of the service mart (the “movie”). In this way, beside adding expressive power to service mart properties, we also model “real world relationships”, i.e. conceptual elements whose purpose is bridging real world objects. Concepts such as “acts-in” between “actor” and “movie” are not mapped to a service mart, they are instead modeled by repeating groups, by placing actors as a repeating group of movie or movies as a repeating group of actor (or both). This goal is consistent with keeping the search computing service infrastructure as simple as possible, and also with



keeping the connection between the two service marts as simple as possible (a single value-based join, as it will be discussed in Section 3.4). Note that alignment cannot be guaranteed if a relationship is rendered as different repeating groups within distinct service marts which are mapped to distinct data sources, one for “movies” and one for “actors”; service marts do not attempt to reconcile data sources and simply offer a uniform method for their description. Note also that if “acting” takes the relevance of a real-world object, e.g. with contractual details or with the schedule of scenes to be filmed, then it will correspond to a distinct service mart.

### 3.2 Logical Level

After the conceptual definition of service marts, we need to describe the way in which we can effectively perform data access. To this purpose, we introduce a second level of abstraction, the logical level, in which each service mart is associated with one or more specific access patterns. An **access pattern** is a specific signature of the service mart with the characterization of each attribute or sub-attribute (not further distinguished in this section) as either input (I) or output (O), depending on the role that the attribute plays in the service call. In the context of logical databases, an assignment of labels I/O to the attributes of a predicate is called predicate adornment, and thus access patterns can be considered as adorned service marts. Moreover, an output attribute is designed as ranked (R) if the service produces its results in an order which depends on the value of that attribute. To ease service composition, we assume that all ranked attributes return a normalized value within the interval  $[0..1]$ <sup>1</sup>. For example, for the service mart “Movie” we can have the following access patterns:

**Movie<sub>1</sub>**(Title<sup>O</sup>, Director<sup>O</sup>, Score<sup>R</sup>, Year<sup>O</sup>, Genres.Genre<sup>I</sup>, Language<sup>O</sup>, Openings.Country<sup>I</sup>,  
 Openings.Date<sup>I</sup>, Actor.Name<sup>O</sup>)  
**Movie<sub>2</sub>**(Title<sup>I</sup>, Director<sup>O</sup>, Score<sup>R</sup>, Year<sup>O</sup>, Genres.Genre<sup>O</sup>, Language<sup>O</sup>, Openings.Country<sup>I</sup>,  
 Openings.Date<sup>I</sup>, Actor.Name<sup>O</sup>)  
**Movie<sub>3</sub>**(Title<sup>O</sup>, Director<sup>I</sup>, Score<sup>R</sup>, Year<sup>O</sup>, Genres.Genre<sup>O</sup>, Language<sup>O</sup>, Openings.Country<sup>I</sup>,  
 Openings.Date<sup>I</sup>, Actor.Name<sup>O</sup>)  
**Movie<sub>4</sub>**(Title<sup>O</sup>, Director<sup>O</sup>, Score<sup>R</sup>, Year<sup>O</sup>, Genres.Genre<sup>O</sup>, Language<sup>O</sup>, Openings.Country<sup>I</sup>,  
 Openings.Date<sup>I</sup>, Actor.Name<sup>I</sup>)

In all cases, “Score” is an output attribute (ranging in  $[0..1]$ ) used for ranking movies, which are presented in descending order of their score, i.e. with highest score movies first. The openings “Country” and “Date” are input parameters, which are used in order to extract movies which are shown in a specific country after a specific opening date (thus enabling the extraction of the most recent movies for that country). Then, in the first access pattern, movies are retrieved by providing as input also one of their genres<sup>2</sup>. In the second access pattern, movies are retrieved by providing as input also

---

<sup>1</sup> We also consider the possibility of service interfaces providing two or more ranking attributes, in such case the service definition includes an aggregation function which indicates how to obtain a score in the  $[0..1]$  interval as a function of the ranking attributes; such score is opaque, whereas the ranking attributes take explicit values within the  $[0..1]$  interval.

<sup>2</sup> Note that there are many genders for each movie, and the query language introduced in Chapter 10 supports retrieving a movie if one gender within the movies’ gender set matches with the input provided to the service.

the title (thus modeling request “search recent films by title”). In the third access pattern, movies are retrieved by providing as input the movie’s director. Finally, the last access pattern allows retrieving all those movies in which a specific actor has played.

Access patterns provide predetermined ways in which users can interact with a service mart and combine service marts in a given search process; they must cover the needs of recurrent user’s queries, guaranteeing that these queries will be efficiently answered. The choice of access patterns represents a limitation on the way in which one can obtain data from the service mart; this limitation is in many times imposed by the existing implementations of the service mart, or necessary in order to guarantee good performance of the retrieval processes. Therefore, defining access patterns requires both a top-down process (from query requirements) and a bottom-up process (from concrete service implementations). In general, this tension between top-down and bottom-up processes is typical of Web service design.

### 3.3 Physical Level

At the physical level of service marts we model **service interfaces**, where each service interface is mapped to concrete data sources; a service interface is a triple including a name, a given access pattern, and an endpoint. For instance, two service interfaces can share the access pattern  $Movie_1$  and describe international and Italian movies, respectively, being available at two different endpoints of the same host machine:

**Movie<sub>11</sub>** (“International Movies”,  $Movie_1$ , <http://www.host.endpoint1>)  
**Movie<sub>12</sub>** (“Italian Movies”,  $Movie_1$ , <http://www.host.endpoint2>)

A service interface may not support some of the attributes of the service mart, e.g., because one source could miss the properties “director” and “source”; this provision allows for a minimal amount of inconsistency between service interfaces, that reflect data availability in sources, and service marts, which must mediate among query requirements and data availability at multiple sources. Unsupported properties are indicated when the service interface is registered.

At a physical level, every service mart is mapped to a **normalized schema**, which is adopted for the relational storage of service mart tuples. The normalized schema includes one primary table for each service mart, storing one row for every real-world object, and then one auxiliary table for each repeating group of the service mart, storing one row for each value of such group. One artificial *key* is created for each service mart row and used as foreign key in every auxiliary table, whose key includes as well a *progressive index*. Normalized tables for the service mart Concert are:

**CONCERT** (Key, ConcertNumber, Theatre, City, Director, Orchestra)  
**SHOW** (Key, Num, Date, Time)  
**PROGRAM** (Key, Num, Composer)  
**PRICE-LIST** (Key, Num, SeatType, Price))

Normalized tables for the service mart Movie are:

**MOVIE** (Key, Title, Director, Score, Year, Language)

**GENRE** (Key, Num, Genre)

**OPENING** (Key, Num, Country, Date)

**ACTOR** (Key, Num, Name)

Results returned by a call to a service interface expose an **interchange format** written in JSON (JavaScript Object Notation) [23], a lightweight data-interchange format easy to read and write by humans and easy to parse and generate by machines. The format descends directly from the conceptual description of the service mart. Below is a JSON Movie instance:

```
{
  "title": "Highlander",
  "director": "Russell Mulcahy",
  "score": "0.7",
  "year": "1986",
  "genres": [
    {
      "genre": "action"
    }
  ],
  "openings": [
    {
      "country": "US",
      "date": "31-10-1986"
    }
  ],
  "actors": [
    {
      "name": "Christopher Lambert"
    },
    {
      "name": "Sean Connery"
    }
  ]
}
```

Note that both the normalized schema and the exchange format depend just on the service mart definition; therefore, all service implementations of the same service mart share the same normalized schema and interchange format.

A **service interface** is a unit of invocation and as such must be described not only by its conceptual schema or logical adornment, but also by its physical properties. There are a huge number of options for characterizing data-intensive services, both in terms of performance and quality. In this chapter (and in the first implementations of service marts) we have focused upon few properties, which can effectively be used

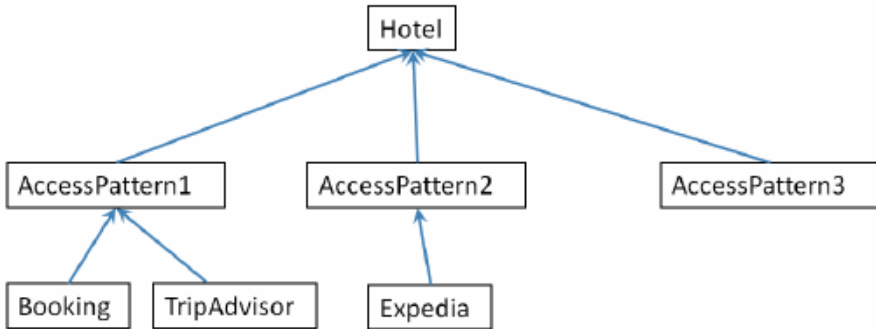
for compile-time optimization (see Chapter 10) and run-time adaptation (see Chapter 12). Service interfaces are described by four kinds of parameters:

- **Ranking descriptors** classify the service interface as a *search service* (i.e., one producing ranked result) or an *exact service*, i.e., services producing objects which are not ranked. Exact services are associated with a *selectivity*, which is a positive number expressing the average number of tuples produced by each call; if the selectivity is within [0..1] the service is denoted as *selective*, otherwise it is called *prolific*. When a search service is associated with an access pattern having one or more output attributes tagged as R, then the ranking is said explicit, else it is said opaque. Explicit ranking over a single attribute can be denoted as ascending or descending. Note that search services may not be present a result with ranked attributes; e.g., most commercial search engines can be characterized as service marts accepting input keywords and producing semi-structured output information which is mapped to a schematic representation, but they normally do not expose rankings in output.
- **Chunk descriptors** deal with output production by a service interface. The service is *chunked* when it can be repeatedly invoked and at each invocation a new set of objects are returned, typically in a fixed number, so as to enable the progressive retrieval of all the objects in the result; in such case, it exposes a *chunk size* (number of tuples in the chunk). Search Computing is focused on the efficient data-intensive computation and therefore most service interfaces are chunked.<sup>3</sup> Of course, if the service is ranked, then the first chunk contains the objects with highest ranking, and subsequent chunks yields the next objects in the ranking; normally, with exact services a query should examine all chunks, while with search services a query can examine just the top chunks.
- **Cache descriptors** deal with repeated invocations of the service. A very efficient way to speed up service invocations consists in caching at the requester side the responses returned for given inputs, and then use such stored answers instead of invoking the service. But such policy is not acceptable with many services, e.g. those offering real-time answers. Hence, parameters indicate if a service interface is *cacheable* and in such case what is the cache *decay*, i.e. the elapsed time between two calls at the source that make the use of stored answers tolerable.
- **Cost descriptors** deal with associating each service call with a cost characterization; this in turn can be expressed as the *response time* (time required in order to complete a request-response cycle), and/or as the *monetary cost* associated with making a specific query (for those systems who charge their answers). Normally, the response time is the most relevant cost factor, because a rapid production of results is crucial.

As a summary of the conceptual, logical, and physical representation of service marts, consider Figure 1, where the service mart about hotels is initially provided with three

---

<sup>3</sup> Chunks enable iterative processing which is similar, in the context of SOA, to cursor-based data retrieval in data management.



**Fig. 1.** Service mart at the conceptual, logical, and physical levels of abstraction

access patterns, and then two of such access patterns are provided with service interfaces, which are then named after the specific Web applications used for implementing the interface. Note that two service interfaces are offered for the first access pattern. In general, each service mart presents several patterns and these in turn present several interfaces, all interfaces share the same schema, and all interfaces for the same access pattern share the same schema adornment.

### 3.4 Connection Patterns

**Connection patterns** introduce a pair-wise coupling of service marts (at the conceptual level) and of service interfaces (at the physical level). Every pattern has a *conceptual name* and then a *logical specification*, consisting of a sequence of simple comparison predicates between pairs of attributes or sub-attributes of the two services, which are interpreted as a conjunctive Boolean expression, and therefore can be implemented by joining the results returned by calling service implementations.

For exemplifying connections, let us assume that two service marts are available and denote hotels and roundtrip selections of pairs of flights, with the schema:

**Hotel**(Name, City, Stars, Score, PriceList(RoomType, Period, Cost), Availability(RoomType, StartDay, EndDay))

**RoundTrip**(HomeCity, DestinationCity, FlightNum1, Date1, TimeDep1, TimeArr1, FlightNum2, Date2, TimeDep2, TimeArr2, PriceList(Fare, Cost))

Assume that services are ranked, e.g. by price in the case of hotels and by fare in the case of roundtrip flights (scoring functions for queries are described in the next chapter). Let us assume that the Availability repeating group gives, for each room type, time intervals when a room is available and therefore can be booked.

These service marts can be linked by several connections, each one with a different semantic meaning. The most useful ones are a time-independent connection, checking just the existence of hotels with certain properties in the destination city, or a time-dependent connection, checking also for room availability. The former is easier to compute and can be used to have a first idea of possible travel compositions, the

second one is more time consuming and can be used when the travel options are better defined. The two connections are specified as follows:

**ExistsHotel(Hotel,RoundTrip): [(City=DestinationCity)]**

**ExistsRoom(Hotel,RoundTrip): [(City=DestinationCity) and (Date1>StartDay) and (Date2<EndDay)]**

Connection pattern have conceptual, logical, and physical properties.

- Conceptually, the two service marts expressing the concepts “Hotel” and “RoundTrip” are therefore connected by two connection patterns, one labelled “ExistsHotel” and the other one labelled “ExistsRoom”.
- Logically, the connections are further specified as a sequence of predicate expressions.
- Physically, some predicates may not be supported by specific service interfaces, e.g. because they do not implement the join attributes.

Visually, service marts and connection patterns can be presented as **resource graphs**, where nodes represent marts and undirected arcs represents connection patterns. Thus, the Search Computing model of the Web is based upon a simplification of reality, which is seen through potentially very large resource graphs. Such representation enables the selection of interconnected concepts that support the creation and dynamic extension of multi-domain queries.

User interfaces for Search Computing allow building queries from resource graphs, thereby specifying only those queries which can be supported. Logical connections completely specify the connection semantics and give users the possibility to build queries, which become supported by a well-defined query language (formally defined in the next chapter).

## 4 Service Implementation

Every service interface must be coupled to a service implementation, which extracts information from the data sources and transforms their format so as to adhere to a standard description in JSON, used to communicate the results of the call. We distinguish three typical scenarios:

- Data can be queried by means of a Web service.
- Data are available on the Web but must be extracted from Web sites through wrappers.
- Data are not directly accessible and must first be materialized.

### 4.1 Web Service Registration

The typical service implementation is a real Web service registered in the platform. Web services return their output in arbitrary format, including but not limited to HTML, XML and JSON. Given that the service mart interchange format is a well-defined JSON structure, the service implementation developer must define a series of

transformations on the results, and bundle them into a remote service implementation that hides the peculiar features of each remote source.

In the case of REST services the transformation is immediate, since the service already return JSON fragment. The typical transformation we need to apply is a selection of the data returned by the service and eventually a transformation in the JSON structure. Also in the case of WSDL services, we have only to define the binding between the service mart and the operation we want to invoke, connecting the appropriate parameters returned by the service to the service mart. Moreover, the service output has to be transformed into an appropriate JSON fragment.

Once the services are transformed to return JSON, another step can be necessary in order to adapt the cardinality of the results returned in each service, which can be not appropriate (a search service could return too many results with each call, or even all the results together). In this case, a *chunker module* supports changing the chunk size: every call to the actual service is translated into the appropriate number of calls to the service implementation, which buffers results and produces chunks of the desired size. Chunker modules are also available in the execution engine (see Chapter 13).

## 4.2 Web Page Wrapping

The second types of sources we want to use are HTML pages. The Web is rich of good quality information stored in HTML Web sites. Wrappers are particular programs that can make available data published in the Web. In the context of service marts, wrappers can be used to capture data which is published by Web servers in HTML format, because in such case a data conversion is needed in order to support data source integration – data must be rearranged according to the service mart normalized schema. Another typical use of wrappers in Search Computing occurs when services respond with HTML documents which must be translated in the normal schema and encoded in JSON.

For building wrappers, several systems are available; in particular we use Lixto [5,14], which is extensively described in Chapter 6. The wrapping process in Lixto relies on one main operation performed by users: by marking a region of an example Web document displayed on screen, using an input device such as a mouse, the user helps the tool to build a set of rules describing the structure of the pages of the Web site. These rules are used by Lixto to generate a wrapper that can be used both to materialize as structured data the entire content of the Web site and to build a dynamic program capable to "query" Web site in real time. Fig. 2 shows the relationships between the data extracted by a query on the Web and a tabular view on these data.

## 4.3 Data Materialization

Even if most service implementations require a *call to a remote service*, in some cases summarized and materialized data may need to be stored at the engine site. In this section, we describe the process that leads to producing service results in a format which is coherent with the service mart organization. Data materialization is a general process, which can be applied to sources in order to transform their format, to eliminate redundancy, to improve their quality, and so on; thus, data materialization

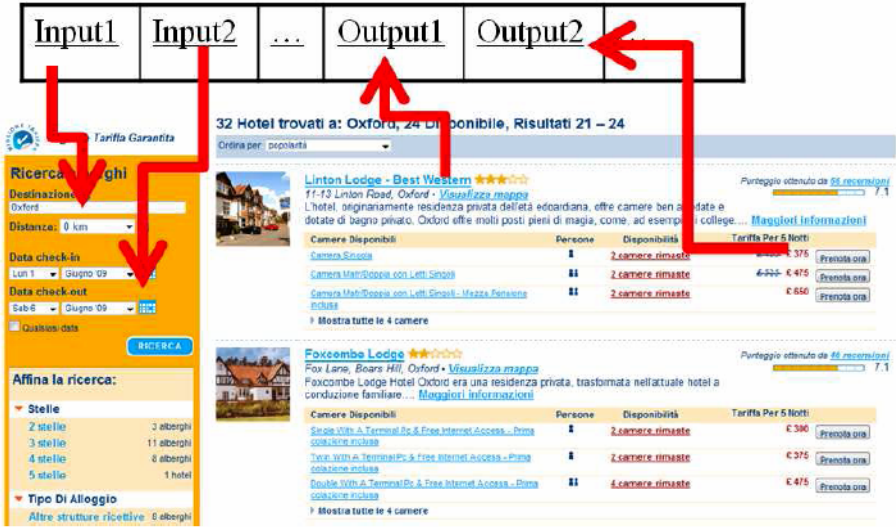


Fig. 2. Data extraction from query results published in HTML

can be performed both on the local (engine) and remote (provider) sites; in the latter case, data is kept at the provider’s site, and retrieved through a dynamic service implementation at query execution. In both cases, data materialization moves data preparation from query execution to source registration time, together with a data materialization schedules setting the times when materialization should be repeated; therefore, data materialization is very useful for supporting efficient query execution. Intrinsic to the normalization process, however, is the capturing of a given snapshot of the data, which is not current; therefore the approach can be used only with data which rarely changes.

We developed a **materializer** specifically for use in Search Computing. The materializer is a software component whose objective is to read arbitrary data sources and organize data in a normalized format, suitable for data export according to a service mart definition. A materializer is organized with two logical layers, shown in Fig. 3.

- The *data extraction layer* operates directly upon the data sources, that can be of arbitrary formats (e.g., tables, XSL files, XML trees, and so on); its purpose is to transform the input data into relational tables of arbitrary format, called primary materialization; such table are temporary, used only in the materializer, and invisible to the outside environment. The transformation primitives of the data extraction layer are implemented as Java components and arranged as a dataflow, which progressively apply to input data; the last primitive stores java objects as table rows.
- The *data formatting layer* uses a series of SQL procedures, applied to the primary materialization, to produce the normalized schema of the service mart, as defined in Section 3.3.



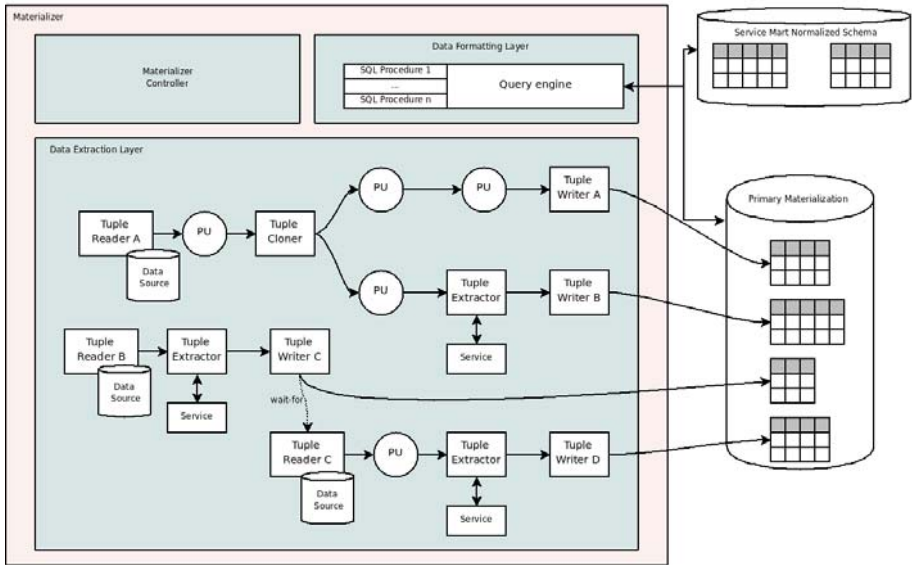


Fig. 3. Process description within the Materializer

Note that data providers need not use the materializer, as long as they build tables according to the normalized schema. These data are published (either locally or remote) and exposed to simple *query modules*, described in the next section, which execute at query time, performing data selection (according to query input), chunking, and linearization in JSON format.

There are two types of processing units: data conversion processing unit and data extraction processing units (specific to the data extraction layer).

The **data conversion processing unit** of the data extraction layer accepts one tuple in input, perform transformations on it, and return a list of output tuples. This unit is programmed by the designer of each service interface and it is specific of each concrete task. This unit is concerned with removing or reordering tuple attributes, with composing new values from existing values (e.g. by extracting different attributes driven by separator characters, or by inserting separation characters when multiple attributes are merged into one), with string manipulation and substitutions. A data conversion processing unit can also be built for building service interfaces, e.g. when data are natively extracted by invoking remote services and then must be transformed so as to fit the standard JSON format.

The processing units of the data extraction layer are:

- **Tuple Reader**, with no input, specialized by data source type, reads a list of data items from the data source and writes them on an output connection;
- **Tuple Writer**, with no output, writes data items as rows in a database table;
- **Tuple Cloner**, with one input and two or more outputs, replicates the data item in input to all its outputs.
- **Tuple Extractor**, with exactly one input and output, invokes a remote data source; every call result generates, from an input item, a list of output items.

The extractor unit enables inserting in the data materialization flow calls to external services, including data wrappers (discussed next). In order to synchronize the components, it is possible to define wait-for relationships, when some components have to wait the production of all the items of a predecessor component before starting their execution.

When data materializations are stored according to the normalized schema, the service implementation can be automatically built by using SQL-based queries whose code depends only on the service interface description (in particular, from its access pattern, i.e. the characterization of fields as input, output, and ranking).

Specifically, queries over stored normalized tables perform selection based upon user input and ranking using the ORDER BY clause. Nested content from the various secondary tables is extracted by using nested queries. While selection, ranking and nesting are supported by standard SQL, chunking requires returning at each call the “top k” tuples, and unfortunately “top k” queries are not supported in standard SQL. Thus, the SQL-based implementation must use a vendor-specific SQL dialect. The general technique is to extract the first key result using the dialect specific feature and build the JSON fragment for these k results.

All commercial DBMS offer “top-k” queries; some of them offer as well “interval” queries, enabling the extraction not only of the “best k”, but also of the “subsequent k” (defined within the interval  $[k+1..2k]$ ), and so on; we show this case. MySQL offers an “interval” query through a LIMIT clause, which returns at each query evaluation the values within an ordered table included between its first and second parameter. Results are then linearized and parsed into JSON. If we use such feature, a simple query pattern for extracting the first k tuples (where k is the chunk size) is:

```
SELECT *
FROM Table
WHERE condition
ORDER BY rank DESC
LIMIT k
```

In order to extract the N-th chunks we have to modify the limit condition using the clause LIMIT p, q, where  $p=k(N-1)$  and  $q=kN$ .

If data contains repeating groups the scenario is a slightly more complicated. We first extract into RESULT-P the tuples of the Primary table which satisfy conditions on either the Primary or the Secondary tables.

```
SELECT *
FROM Primary
WHERE conditionOnPrimary
AND Key IN
  (SELECT KEY
   FROM Secondary1
   WHERE ConditionOnSecondary1)
UNION ...
  (SELECT KEY
   FROM SecondaryN
   WHERE ConditionOnSecondaryN)
```

```
ORDER BY rank DESC
LIMIT k
```

We then extract into RESULT-I the tuples from the *i*-th Secondary tables with matching keys with the table RESULT.

```
SELECT *
FROM Secondary-i
WHERE Key IN SELECT Key
FROM R
```

The tables in RESULT and RESULT-S are then used for building the JSON interchange format; the parser uses the key to combine tables. Such process is iterated for the various chunks, by changing the LIMIT clause in the RESULT-P query.

In most cases, queries require interacting with remote services. In this case we use the modules capable to manage direct service calls inside the materialize. Two additional modules can be very useful for manipulating the results retrieved from remote services: a *group-by module* supporting the construction of rows for auxiliary tables by grouping the distinct values of rows which have certain values in common and the *chunker module* already described.

## 5 Running Example

In this section we describe the service marts supporting a running example in common to the chapters of this part: *find a good recent adventure movie in a theatre not too far from home and then find a good restaurant in walking distance*. This query initially requires the integration of three domains: movies, theatres, and restaurants. Given that the services we use are focused on movies in English, we then formulate the query in US, e.g. “Marina in San Francisco”, “University Avenue in Palo Alto”, or “Washington Square in New York”.

### 5.1 Service Mart: “Movie”

The running example requires a search service capable of filtering movies by time (e.g., whose opening date in US is recent enough) and genre (e.g. action movies) and then extracting them ranked by their quality score. For such purpose, we use the IMDB archive (<http://www.imdb.com>), which stores information about thousands of movies and enriches their description with a “score” attribute computed as the average of the scores assigned by community users worldwide to movies.

IMDB allows two different interaction modes: direct access to Web pages, or usage of Web services. One of them (<http://www.trynt.com/trynt-movie-imdb-api>) provides a programmatic access to a sub-set of IMDB data, featuring multiple export formats (including JSON), and returning several data fields including actors, titles, evaluators, and their comments. Unfortunately, the web service lacks a method that allows one to query the service to obtain the movies filtered by genre and ordered by their quality. Therefore, IMDB is managed by building an ad-hoc wrapper and using it to materialize all movie descriptions and putting them into the “Movie” normal schema. This requires periodic downloads to maintain such data materialization

up-to-date; weekly updates to capture new openings seem adequate. For the running example, we use the service interface `Movie11`,

**Movie<sub>11</sub>** (“**International Movies**”, **Movie<sub>1</sub>**, **http://www.host.endpoint1**)

with access pattern `Movie1` having inputs on the “Genre” of the movie and on the “Country” and “Date” of opening, and ranking on the “Score”:

**Movie<sub>1</sub>**(**Title<sup>0</sup>**, **Director<sup>0</sup>**, **Score<sup>R</sup>**, **Year<sup>0</sup>**, **Genres.Genre<sup>I</sup>**, **Language<sup>0</sup>**, **Openings.Country<sup>I</sup>**, **Openings.Date<sup>I</sup>**, **Actor.Name<sup>0</sup>**)

Data extraction, in turn, requires materializing four tables, as discussed in Section 3:

**MOVIE** (**Key**, **Title**, **Director**, **Score**, **Year**, **Language**)

**GENRES** (**Key**, **Num**, **Genre**)

**OPENINGS**(**Key**, **Num**, **Country**, **Date**)

**ACTOR** (**Key**, **Num**, **Name**)

In order to return the first ten results we use the MySQL query below, which fills the `RESULT` table:

```
SELECT *
FROM Movie
WHERE Date > 2008 AND key IN
    (SELECT key
     FROM Genres
     WHERE Genre=INPUT1)
AND Key IN
    (SELECT key
     FROM Openings
     WHERE Country=INPUT2 and Date>INPUT3)
ORDER BY Score DESC
LIMIT 0,10
```

Three additional queries are used to extract genres, actors and openings:

```
SELECT *
FROM Actors
WHERE Key IN SELECT Key
              FROM Result
```

```
SELECT *
FROM Genres
WHERE Key IN SELECT Key
              FROM Result
```

```
SELECT *
FROM Openings
WHERE Key IN SELECT Key
              FROM Result
```

Results returned by this query are used by a parser which builds one JSON fragment for each film, within all its openings, genres, and actors.

If we want instead to retrieve Italian movies, we can use an Italian Web site, <http://www.hyperreview.com>, which lists movies with Italian titles; therefore, a second service interface is developed for the same service mart, which extracts the data using the same materialization mechanism.

## 5.2 Service Mart: “Theatre”

The next step in our example is the registration of a service capable to offer a list of movie theatres, with the related films, ordered w.r.t. the distance from a given location. We define a service mart Theatre:

**Theatre**(Name, UAddress, UCity, UCountry, TAddress, TCity, TCountry, TPhone, Distance, Movie(Title, StartTimes, Duration))

U versions of attributes “Address”, “City” and “Country” denote the user’s location, while T versions of the same attributes represent the theatre location; Theatre is connected to Movie via a connection pattern “Shows” using a join on titles:

**Shows**(Movie, Theatre): [(Title=Title)]

For the service implementation, we use “*Movie Showtimes - Google Search*” (<http://www.google.com/movies>), a service allowing the retrieval of all the cinemas nearby an input location that is expressed as a complete address (address, city, country) or as a city. We select the second case and therefore the service interface:

**Theatre**<sub>1</sub>(Name<sup>0</sup>, UAddress<sup>I</sup>, UCity<sup>I</sup>, UCountry<sup>I</sup>, TAddress<sup>0</sup>, TCity<sup>0</sup>, TCountry<sup>0</sup>, TPhone<sup>0</sup>, Distance<sup>R</sup>, Movie.Title<sup>0</sup>, Movie.StartTimes<sup>0</sup>, Movie.Duration<sup>0</sup>)

The service returns results sorted by theatre distance from the input location, but it does not return the actual distance (therefore, ranking is opaque, and the implementation does not expose “Distance”). Moreover, we have noticed that the sorting is approximate, probably because theatres are selected by Google based on their importance (number of parallel shows). Therefore, we have developed a second service interfaces which integrates the Yahoo! Maps Web Service, that allows one to find the specific latitude and longitude for an address, and then computes the exact distance between two points (home and theatre) so as to compute explicitly the distances of theatres, expose them in the output, and then re-rank outputs based upon the exact distance. The second service implementation demands more resources and requires longer computations (but it exposes “Distance”). Then, the two service interfaces are registered and mapped to their endpoints:

**Theatre**<sub>11</sub> (“Google Service”, Theatre<sub>1</sub>, <http://www.host.endpoint1>)

**Theatre**<sub>12</sub> (“Exact Google Service”, Theatre<sub>1</sub>, <http://www.host.endpoint2>)

An example of instance extracted from Theatre<sub>11</sub> is:

```
{
  "Name" : "Pacific Culver Stadium 12",
  "TAddress" : "9500 Culver Boulevard",
  "TCity" : " Culver City",
  "TCountry" : " CA, USA ",
  "TPhone" : " (310) 360-9565 ",
  "Movies" : [
    {
      "Title" : "Law Abiding Citizen",
      "Duration" : "1hr 48min ",
      "StartTimes" : "12:05am"
    },
    {
      "Title" : "Paranormal Activity",
      "Duration" : "1hr 39min ",
      "StartTimes" : "12:05am"
    }
  ]
}
```

In order to retrieve Italian theatres, we use <http://www.paginegialle.it/>, a Web site listing the commercial activities belonging to a specific category (such as restaurant, bar, theatres etc) that are located in or nearby a specific geographical place. Therefore we build a third service interface Theatre<sub>13</sub> covering the Italian theatres over “Pagine Gialle”. Since this service does not provide a set of API for the data extraction, it is wrapped by using Lixto.

### 5.3 Service Mart: “Restaurant”

In our running example, once we decide about the theatre, we then look for a walking-distance restaurant. We introduce the service mart:

**Restaurant(Name, UAddress, UCity, UCountry, RAddress, RCity, RCountry, Phone, Url, MapUrl, Rating, Distance, Category(Name))**

Then we qualify attributes “UAddress”, “UCity”, “UCountry”, and “Category.Name” as I (corresponding to the starting location and the kind of restaurant we look for), “RAddress”, “RCity”, “RCountry”, “Location”, “Phone”, “Url” and “MapURL” as O (for locating and inspecting the restaurant) , “Distance” and “Ranking” as R (for ranking a combination of quality and distance), yielding to the following access pattern:

**Restaurant<sub>1</sub> (Name<sup>O</sup>, UAddress<sup>I</sup>, UCity<sup>I</sup>, UCountry<sup>I</sup>, RAddress<sup>O</sup>, RCity<sup>O</sup>, RCountry<sup>O</sup>, Phone<sup>O</sup>, Url<sup>O</sup>, MapUrl<sup>O</sup>, Distance<sup>R</sup>, Rating<sup>R</sup>, Category.Name<sup>I</sup>)**

The service marts Theatre and Restaurant have a connection pattern “Dinner Place” specified as:

**DinnerPlace(Theatre, Restaurant): [(TAddress=UAddress),  
(TCity=UCity), (TCountry=UCountry)]**

For the implementation of the service interface, we use the Yahoo Local source (<http://local.yahoo.com/>), a service that allow the users to find Businesses & Commercial Services (e.g. restaurants) that are in or nearby a requested address, city and state, or a specific zip code. In order to access to the Yahoo Local data, we use the Yahoo! Query Language (YQL) Web Service, which enables the access of Internet data with SQL-like commands and report the results in XML or JSON output format.

YQL statements can be executed upon Yahoo Local as REST services. Yahoo Local provides commercial exercises located in Australia, Canada, Germany, France, India, Korea, US, and UK, but not in Italy. Therefore, we also consider a service interface of the same service covering Italy, which uses as physical service Yahoo Travel (<http://travel.yahoo.com/>); this service is not included in the YQL Web Services suite, but it has available APIs on the Web.

## 6 Conclusions

This chapter has provided the definition of service marts as an interoperability concept for building Search Computing applications. The Web world is described as a resource graph with service marts linked by and connection patterns, and then service marts are associated with service interfaces and implementations. Service delivery uses well-defined formats and rules so as to enable the execution of queries and the composition of tuples so as to build query results. Services of the running example show that content already exists on the Web, and that data extraction requires selecting sources and designing service mart, with a mix of top-down and bottom-up attitude; however, setting up these services is not too difficult.

## References

1. Al-Masri, E., Mahmoud, Q.H.: Investigating web services on the world wide web. In: Proceeding of the 17th international Conference on World Wide Web, WWW 2008, Beijing, China, April 21-25, pp. 795–804. ACM, New York (2008)
2. Alonso, G., Casati, F., Kuno, H., Machiraju, V.: Web Services: Concepts, Architecture and Applications. Springer, Heidelberg (2004)
3. Ankolenkar, A., Burstein, M., Hobbs, J.R., Lassila, O., Martin, D., McDermott, D., McIlraith, S.A., Narayanan, S., Paolucci, M., Payne, T.R., Sycara, K.: DAML-S: Semantic Markup for Web Services. In: The First International Semantic Web Conference (ISWC), Sardinia (Italy)
4. Austin, D., Barbir, A., Ferris, C., Garg, S. (eds.): Web Service Architecture Requirements. W3C Working Group Notes (2004), <http://www.w3.org/TR/wsa-reqs>
5. Baumgartner, Flesca, S., Gottlob, G.: Visual Web Information Extraction with Lixto. In: Proceedings of the 27th Very Large Data Bases Conference, September 11-14, pp. 119–128 (2001)

6. Bonifati, A., Cattaneo, F., Ceri, S., Fuggetta, A., Paraboschi, S.: Designing data marts for data warehouses. *ACM Trans. Softw. Eng. Methodol.* 10(4), 452–483 (2001)
7. Ceri, S., Matera, M., Rizzo, F., Demaldè, V.: Designing data-intensive web applications for content accessibility using web marts. *Commun. ACM* 50(4), 55–61 (2007)
8. Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., Orchard, D. (eds.): *Web Services Architecture*, <http://www.w3.org/TR/ws-arch/>
9. de Bruijn, J., Fensel, D., Keller, U., Lara, R.: Using the web service modelling ontology to enable semantic eBusiness. *Communications of the ACM (CACM), Special Issue on Semantic eBusiness* (2005)
10. Dietze, S., Gugliotta, A., Domingue, J.: Towards context-aware semantic web service discovery through conceptual situation spaces. In: Sheng, Q.Z., Nambiar, U., Sheth, A.P., Srivastava, B., Maamar, Z., Elnaffar, S. (eds.) *Proceedings of the 2008 international Workshop on Context Enabled Source and Service Selection, integration and Adaptation: Organized with the 17th international World Wide Web Conference (WWW 2008), CSSSIA 2008, Beijing, China, April 22, vol. 292, pp. 1–8. ACM, New York* (2008)
11. Fensel, D., Bussler, C.: The Web Service Modeling Framework WSMF. *Electronic Commerce: Research and Applications* 1(2), 113–137 (2002)
12. Fensel, D., Kerrigan, M., Zaremba, M.: *Implementing Semantic Web Services*. Springer, Heidelberg (2008)
13. Garofalakis, J., Panagis, Y., Sakkopoulos, E., Tsakalidis, A.: Web Service Discovery Mechanisms: Looking for a Needle in a Haystack? In: *International Workshop on Web Engineering* (2004)
14. Gottlob, G., Koch, C., Baumgartner, R., Herzog, M., Flesca, S.: The Lixto data extraction project: back and forth between theory and practice. In: *Proceedings of the Twenty-Third ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2004, Paris, France, June 14 - 16, pp. 1–12. ACM, New York* (2004)
15. Laliwala, Z.: Event-driven Dynamic Web Services Composition and Automation of Business Processes. *Services Computing*. In: *IEEE International Conference on Services Computing (SCC 2006)*, pp. 527–528 (2006)
16. Leymann, F.: *Web Services Flow Language (WSFL 1.0)*, IBM (May 2001)
17. Overhage, S.: On Specifying Web Services Using UDDI Improvements. In: *3rd Annual International Conference on Object-Oriented and Internet-based Technologies, Concepts, and Applications for a Networked World Net.ObjectDays, Germany* (2002)
18. Plebani, P., Pernici, B.: URBE: Web Service Retrieval Based on Similarity Evaluation. *IEEE Transactions on Knowledge and Data Engineering* 21(11), 1629–1642 (2009)
19. Roman, D., Keller, U., Lausen, H., de Bruijn, J., Lara, R., Stollberg, M., Polleres, A., Feier, C., Bussler, C., Fensel, D.: *Web Service Modeling Ontology. Applied Ontology* 1(1), 77–106 (2005)
20. AXIS, <http://ws.apache.org/axis/>
21. CXF, <http://cxf.apache.org/>
22. CORDIS, <http://cordis.europa.eu/fp7/ict/ssai/> (September 17, 2009)
23. JSON, <http://JSON.org/>
24. REMOTEMETHOD, <http://www.remotemethods.com/>
25. RESTFUL, <http://java.sun.com/developer/technicalArticles/WebServices/restful/>
26. SAWSDL, <http://www.w3.org/2002/ws/sawsdl/>
27. SOAP, <http://www.w3.org/TR/soap/>
28. SPRING, <http://www.springsource.org/>



29. StrikeIron, <http://www.strikeiron.com>
30. UDDI,  
[http://www.oasis-open.org/committees/  
tc\\_home.php?wg\\_abbrev=uddi-spec](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=uddi-spec)
31. Woogle, <http://db.cs.washington.edu/webService/>
32. WSDL, <http://www.w3.org/TR/wsdl>
33. Wsoogle, <http://wsoogle.com>
34. XMethods, <http://www.xmethods.com>

# Chapter 10:

## Join Methods and Query Optimization

Daniele Braga, Stefano Ceri, and Michael Grossniklaus

Dipartimento di Elettronica e Informazione, Politecnico di Milano,  
Piazza Leonardo da Vinci 32, 20133 Milano, Italy  
{braga,ceri,grossniklaus}@elet.polimi.it

**Abstract.** Joins between data sources are an essential ingredient of multi-domain queries, as they exploit connection patterns defined between service marts or between service interfaces. This chapter moves from the definition of a query language over service interfaces, sketching how queries can be directly expressed over service marts and how these can be translated over service interfaces. The fundamental operation discussed in this chapter is the binary join between two sources, which is influenced by the type (search vs. exact) of services and by the management (parallel vs. sequential) of service calls. Then, this chapter presents an optimization framework for queries over several service interfaces, which considers several cost metrics for mapping queries into query plans, consisting of specific operations over services, and includes a branch and bound approach to the exploration of the combinatorial search space of all possible query plans.

### 1 Introduction

This chapter delves into the issues of formulating and optimizing multi-domain queries over several services, focusing on the specific problems that arise due to the presence of search services in the queries. The distinguishing feature of a search service is to return answers in relevance order. In general, although the answers produced by a search service can be very numerous, users are only concerned with the answers provided within the first pages of results. Thus, a query strategy that retrieves all the answers from a search service is rarely appropriate. On the other hand, only the user can correctly evaluate the relevance of answers produced by search engines. Therefore, if a query involves several searches, all answers produced by the involved search engines should be composed in the query output and presented to the user for a correct evaluation. Moreover, the user expects answers in ranking order. Thus, while composing results from multiple services, answers should be presented according to a global ranking that is obtained either as an exact composition function of the rankings (then, we can talk about “top-k” results) or an approximation of that function.

In this chapter, we define a **formal model** for the optimization and the execution of multi-domain queries over services which expose heterogeneous information sources. Such model serves as a unifying perspective for several diverse possible application settings, ranging from providing expert users with service mash-up tools to providing the foundations for vertical service integration frameworks. The originality of the model stems from the way in which data sources are classified, distinguishing

between *exact* services, that have a “relational” behavior and return either a single answer or a set of unranked answers, and *search* services, that return a list of answers in ranking order, according to some measure of relevance.

We also formally define **query plans**, playing the same role within our context as physical access plans in relational databases. A plan is defined as the orchestration of service invocations, possibly in parallel, which takes into account some significant features of the service, including its ability to return results in chunks. Query plans schedule the invocations of Web services and the composition of their inputs and outputs. Within plans, the main operations are **joins** between results, whose execution can take place according to several join strategies.

Then, we define an **optimization method** for choosing the “best” access plan, i.e. the one that optimizes an objective function, resorting to a classical and well-grounded approach such as branch and bound in order to efficiently explore the solution space. Cost minimization is performed against one of several alternative *cost metrics*, capturing different scenarios and different optimization goals.

The organization of this chapter is as follows. Section 2 presents the state-of-the-art of query optimization, with a specific emphasis upon systems that integrate data collected through Web services. Section 3 presents the query language, offering its syntax, semantics, and exemplification on the running example. It then presents a graph model for query plans in which nodes denote operations and arcs denote the dataflow between them. The most important operation is the join of two services, and Section 4 focuses on join methods. Finally, Section 5 shows the optimization steps and heuristics leading to the selection of an optimal plan.

## 2 State of the Art

The background for this chapter ranges over several disciplines. In this Section we move from the classical foundations of query processing and optimization, in order of increasing specificity, to Web service composition, query answering under access limitation, and the study of systems dedicated to query optimization over Web services (Web Service Management Systems).

### 2.1 Query Processing Foundations

Query processing is perhaps the fundamental technology offered by database management systems, in the sense that they provide means for translating declarative SQL queries into highly efficient access plans. The main merit of query processing is the ability of producing high performance plans. Work in query processing can be traced back to foundational papers such as [7]. Dedicated books, such as [30], focus just on query processing. In most approaches, query processing consists first in giving an abstract representation of the query (typically in the form of a directed acyclic graph having data resources as leaves and data extraction and transformation operations as intermediate nodes [7]), and then using equivalence transformations upon such graphs in order to use the best possible operations and settings of operations’ parameters, so as to minimize an objective function. Such processing is called query optimization and typically uses methods of operations research

(e.g., [15]). Extensible query processors are obtained by describing equivalence transformations as rules, and then varying the rule set by adding or dropping rules or by varying their priority [17]. Other well-known query optimization techniques exist, such as *transformation*-based approaches [21] or *randomized* approaches [13].

*Branch-and-bound* algorithms are, e.g., adopted in [24], which considers a query optimization problem with characteristics that are similar to our problem. Specifically, the authors focus on ranked queries in the context of classical databases, where the “ranking” is expressed by means of an *explicit* preference function over the values of a tuple’s attributes, to be taken into account when computing top-k answers. However, service characteristics and implicit ranking orders are not dealt with, which instead are a distinguishing feature of the work presented in this chapter. We propose a branch-and-bound query construction method driven by a set of heuristics that allow us to take into account the peculiarities of search services and to converge to an optimal solution with respect to a given cost metric. Like in [12], in our case we cannot easily resort to transformation-based techniques, as the presence of access patterns and ranking orders does not guarantee properties like associativity and commutativity for joins. Randomized approaches, on the other hand, are not efficiently applicable with explicit access patterns, as this would typically lead to uselessly consider a large number of infeasible plans during query optimization.

## 2.2 Answering Queries over Web Services

Historically, answering queries over independent data sources has been the research object of *parallel* or *distributed query processing* [14][23][11]. Two main techniques have emerged in this research field: code shipping and data shipping. While code shipping to Web services is not feasible, data shipping is feasible and allows the feeding of results coming from one service in the access plan to another service in the plan. The latter technique is heavily leveraged in our work, as data are shipped in pipelines from one service to another, so as to maximize parallelism.

The coordinated execution of distributed Web services is the subject of *Web services composition*, which comes in two different flavors: orchestration and choreography [9]. The distributed approach of *choreographed* services (e.g., using WS-CDL [26] or WSCI [27]) does not suit our query processing problem, because choreographies are not executable and require the awareness of and compliance with the choreography by all the involved services. The centralized approach of *orchestrated* services (e.g., using BPEL [19]) suits better the research problem addressed in this paper, as orchestrations are executable service compositions (i.e., query plans, in our terminology) and services need not be aware of being the object of query optimization and execution. In the specific case of BPEL, however, its workflow-based approach does not provide the necessary flexibility when the invocation order of services needs to be computed at runtime, as is our case (e.g., dynamically fixing a number of fetches to be issued to a service remains hard). There is a growing amount of semantic approaches to the runtime composition of Web services (e.g., [28] or [8]), but their focus is typically on functional requirements or quality of service [2] and less on data.

Inspired by the work presented in [22], Tatemura et al. [25] introduce the idea of continuous query over service-provided data feeds (e.g., through RSS or Atom). The

goal is to mash up and monitor the evolution of third-party feeds and to query the obtained result. Their mash-up query model is articulated into collections of data items and collection-based streams of data (streams also track the temporal aspect of collections and allow the querying of the history of collections). Suitable select, join, map, and sort operators are provided for the two constructs. The described system consists of a visual mash-up composer, an execution engine, and interfaces for users to subscribe to mash-up feeds equipped with personalized queries.

Finally, Yahoo Pipes<sup>1</sup> and IBM Damia<sup>2</sup> [1] enable a Web 2.0 approach to compose (“mash up”) queries over distributed data sources like RSS/Atom feeds, comma-separated values, XML files, and similar. Both approaches come with user-friendly and intuitive Web interfaces, which allow users to draw workflow-like data feed logics based on nodes representing data sources, data transformations, operations, or calls to external Web services. Both Pipes and Damia require the user to explicitly specify the query processing logic procedurally, which is generally not a trivial task for unskilled users, especially for the case of joins, which have to be explicitly programmed by the user. Instead, with our approach we automatically derive a plan from a declarative query formulation.

It is worth noting that the previous service querying approaches effectively enable users to distribute a query over multiple Web services, but they do not specifically focus on the peculiarities of search services, such as ranking and chunking. Characteristics like the ranking order of results or advanced querying techniques are not considered.

### 2.3 Answering Queries under Access Limitations

Web sources are not freely accessible as in the traditional relational setting, because they typically expose a limited number of interfaces, in which certain fields must be mandatorily filled in order to obtain a result. These fields may be the input fields of a form on a data-intensive Web site or the input parameters of a Web Service invocation. Such access limitations are crucial to the optimization problem, and modeled by characterizing service parameters as binding patterns, i.e., classifying them as input or output parameters, thereby clarifying the different ways in which it can be invoked. The issue of processing queries under access limitations, by some authors studied under the headline of *binding patterns*, has been widely investigated in the literature [20][16][18][29][10].

In our work, we have assumed that queries are always designed so as to admit at least one choice of access patterns. However, for some queries, it may happen that no permissible choice of access patterns exists. Although, in this case, the original user query cannot be answered, it may still be possible to obtain a subset of the answers to the original user query by invoking services that are not necessarily mentioned in the query, but that are available in the schema. In particular, such “off-query” services may be invoked so that their output fields provide useful bindings for the input fields of the services in the query with the same abstract domain. A query “augmentation” of this kind, however, can only provide an approximation of the original query that, in

---

<sup>1</sup> <http://pipes.yahoo.com/pipes/>

<sup>2</sup> <http://services.alphaworks.ibm.com/damia/>

general, requires the evaluation of a recursive query plan even if the initial query was non-recursive.

The problem of finding all obtainable answers to a query posed over data sources with access limitations has been studied in [18] and later works. It has been shown that, even though a query is conjunctive, finding all obtainable answers in general requires a recursive query plan. Also, since accessing data sources over the Web is typically a costly task, later works have addressed the issue of reducing the accesses to the sources, while still returning all obtainable answers. For instance, some optimizations to be made during query plan generation to minimize the accesses to data sources are discussed in [16] for a subset of conjunctive queries, named *connection queries*. More expressive classes of queries, including conjunctive queries, are covered in [6].

## 2.4 Web Service Management Systems

The paper in which Srivastava et al. [22] introduced the notion of Web Service Management System can be considered as one of the main inspiration sources of the query optimization framework in SeCo. The authors propose a Web service management system (WSMS) that enables querying multiple Web services in a transparent and integrated fashion, similarly to the problem approached in this paper. The authors propose an algorithm for arranging a query's Web service calls into a pipelined execution plan that exploits parallelism among Web services to minimize the query's total running time under a bottleneck cost metric (i.e., by choosing to optimize the execution of the slowest service). They assume all services to be exact and with no chunking of results, and model them by means of their per-tuple response time and selectivity. They focus upon simple queries where all input attributes get their values from either exactly one other Web service or from the user's input, and did not consider the peculiarity of search services, with ranked results.

Inspired by [22], Braga et al. developed in [4] the foundations of the optimization framework for Search Computing that is here addressed in Section 5.

## 3 Query Formulation and Translation into an Executable Plan

The optimization of a query over a set of service marts starts from a formulation of the query in a conjunctive query language and ends with a fully instantiated invocation schedule. The execution environment to which the invocation schedule is addressed is a system capable of executing query plans (as they will be formally defined in the sequel). This means that the system can execute requests, collect their results, and integrate them progressively, forming the answers as combinations of partial invocation results.

### 3.1 Query Formulation

We consider select-join queries on service marts and connection patterns. Our formalism abstracts away from the details of any underlying representation, and for each information source resorts to simple service definitions as illustrated in Chapter 9. No projection is performed, as all the data of a service implementation are

presented to the liquid query interface, where the format of results includes projecting over some of the service mart attributes. The user interface may also present one copy of all attributes which are set equal by a query. Queries can be expressed, with exactly the same syntax and semantics, either over service marts or over service interfaces. In the former case, the query processor must select suitable service interfaces so as to make the query *feasible*, according to the definition given below. In this chapter (and in our first conceptualization of Search Computing) we assume that users operate directly over service interfaces. Hence, the interface selection process, though mentioned later on with respect to query optimization, is not part of the query processing chain.

More formally, a **query** consists of a set of services  $s_1, \dots, s_n$  (the same service can occur several times with a different renaming for each different use), a set of selection predicates, and a set of join predicates.

We recall from the previous chapter that an attribute of a service can be either an atomic attribute or a repeating group. A repeating group consists of a non-empty set of atomic sub-attributes that collectively define one property of an object. Atomic attributes are single-valued, while repeating groups are multi-valued. We indicate an attribute  $A$  of a service  $s$  as  $s.A$ . A sub-attribute  $A$  of a repeating group  $s.R$  is indicated as  $s.R.A$ . If no ambiguity arises, the prefixes  $s$  or  $s.R$  may be omitted.

A selection predicate is an expression of the form  $A \text{ op } const$ , where  $A$  is an atomic attribute or sub-attribute,  $const$  is a type-compatible constant, and  $op$  is a comparator among  $\{=, <, <=, >, >=, \text{like}\}$ . A join predicate is an expression of the form  $A \text{ op } B$ , where  $A$  and  $B$  are type-compatible attributes or sub-attributes, and  $op$  is a comparator among  $\{=, <, <=, >, >=, \text{like}\}$ .

A service  $s$  from a query is **reachable** if, for every input (sub-)attribute  $A$  of  $s$ , the query contains a selection predicate of the form  $A=const$ , or a join predicate of the form  $A = B$  where  $B$  is a (sub-)attribute of a reachable service. A query is **feasible** if all its services are reachable.

A tuple of a service is a mapping that sends each attribute  $s.A$  into a value of the domain of  $A$ . For a tuple  $t$  of  $s$ , we use the notation  $t.A$  to indicate the value of  $t$  for attribute  $s.A$ . Note that, if  $s.R$  is a repeating group, the value  $t.R$  is a set of tuples over the sub-attributes of  $s.R$ .

The **semantics** of a feasible query over  $s_1, \dots, s_n$  with a set  $P$  of (selection and join) predicates is defined as the largest set of composite tuples of the form  $t_1 \cdot \dots \cdot t_n$  such that the following two conditions hold:

1.  $t_i \in s_i$  for  $1 \leq i \leq n$ ; and
2. there is a mapping  $M$  from each repeating group of the form  $s_i.R$  occurring in  $P$  into a tuple  $M(s_i.R)$  in  $t_i.R$  such that each expression in the set obtained from  $P$  by
  - replacing each occurrence of  $s_i.R$  with  $M(s_i.R)$  and, after that,
  - replacing each occurrence of  $s_i$  with  $t_i$

is satisfied according to the natural interpretation of comparators.

Consider two services  $S_1$  and  $S_2$  over the repeating group  $R$  with sub-attributes  $A$  and  $B$ . Assume that  $S_1$  provides two objects  $t_1 = (\{<1,x>, <2,x>\})$ ,  $t_2 = (\{<2,x>, <1,y>\})$

and that  $S_2$  provides two objects  $t_3 = (\langle 1, x \rangle, \langle 2, y \rangle)$ ,  $t_4 = (\langle 2, x \rangle)$ . Thanks to the above choice of semantics, the query  $Q_1$ : *select*  $S_1$  *where*  $S_1.R.A=1$  *and*  $S_1.R.B=x$  produces the result  $\{t_1\}$ , and the query  $Q_2$ : *select*  $S_1, S_2$  *where*  $S_1.R.A=S_2.R.A$  *and*  $S_1.R.B=S_2.R.B$  produces the result  $\{t_1 \cdot t_3, t_1 \cdot t_4, t_2 \cdot t_4\}$ . Note that  $t_1$  belongs to  $Q_1$ 's result because  $S_1.R$  can be replaced by the tuple  $\langle 1, x \rangle$  of  $t_1.R$  and the resulting expressions  $1=1$  and  $x=x$  are trivially satisfied. Informally,  $t_1$  is selected because one of its repeating groups satisfies the selection condition. Conversely,  $t_2$  does not belong to  $Q_1$ 's result because, although its sub-attributes separately satisfy the selection, this occurs in different tuples of the repeating group. Therefore no individual tuple of the repeating group satisfies the selection condition. Similarly, note that the tuple  $t_2 \cdot t_3$  does not belong to  $Q_2$ 's result because, although its sub-attributes satisfy the join condition, this occurs in different tuples of the repeating group.

Instead of constants in the query we may also use variables prefixed as INPUT, whose value is provided by users at query execution time. Using the above syntax and semantics, the example query can be expressed as follows:

***RunningExample:***

```
Select Movie11 As M, Theatre11 as T, Restaurant11 as R
where
(selection conditions)
  M.Genres.Genre=INPUT1 and M.Openings.Country=INPUT2 and
  M.Openings.Date>INPUT3 and T.UAddress=INPUT4 and T.UCity=INPUT5
  and T.TCountry=INPUT2 and T.Category.Name=INPUT6 and
(join conditions)
  M.Title=T.Title and T.TAddress=R.RAddress and T.TCity=R.RCity
  and T.TCountry=R.RCountry.
```

Note that the condition  $M.Openings.Country=INPUT2$  *and*  $M.Openings.Date>INPUT3$  extracts movies such that a single opening tuple satisfies both the conditions on country and date.

Join predicates used by a query are normally establishing join condition over connection patterns. Therefore, join conditions can be expressed in a more compact way by mentioning connection patterns, yielding to the formulation below:

***RunningExample:***

```
Select Movie11 As M, Theatre11 as T, Restaurant11 as R
where Shows(M,T) and DinnerPlace(T,R) and
  M.Genres.Genre=INPUT1 and M.Openings.Country=INPUT2 and
  M.Openings.Date>INPUT3 and T.UAddress=INPUT4 and T.UCity=INPUT5
  and T.TCountry=INPUT2 and T.Category.Name=INPUT6
```

Checking the feasibility of this query is immediate by considering that all input places of  $Movie_{11}$  and  $Restaurant_{11}$  are associated with INPUT variables, hence they are reachable, and that by virtue of the join variables linking Theatre to Restaurant also Restaurant is reachable. If all services are properly designed and registered, queries over service interfaces whose join conditions include the connection patterns and whose selection conditions include an equality predicate with either a constant or an input variable are feasible queries. Tools for drawing queries upon the graph representation of service marts and connection patterns can help query designers, so as to enable the drawing only of feasible queries [5].



We finally turn to expressing rankings, an important aspect of Search Computing queries. We assume that each service interface  $s_i$  is associated with a **scoring function**  $SF_i$ . If  $s_i$  is ranked,  $SF_i$  indicates how to obtain a score in the  $[0,1]$  interval as a function of the attributes of  $s_i$ <sup>3</sup>; if it is unranked, the  $SF_i$  is a fixed constant. Then, the query is associated with a **ranking function**  $f$  expressed as a sequence  $(w_1, \dots, w_n)$  of non-negative weights for the scores used in the query. If tuples  $t_1, \dots, t_n$  from, respectively,  $s_1, \dots, s_n$  are used to form a combination, the ranking function of the formed combination  $t_1 \cdot \dots \cdot t_n$  is given as  $w_1S_1 + \dots + w_nS_n$ , where  $S_i$  is the score of  $t_i$  for  $1 \leq i \leq n$ ; the weight of unranked services is set equal to 0. In the above query, with 3 ranked service interfaces, a possible ranking function is  $(0.3, 0.5, 0.2)$ .

Ranking functions may be assigned prior to query execution, either at query definition time or at query presentation time. They can also be altered dynamically through the query interface, yielding to changes in the query execution strategy. Only ranking functions defined at query definition time can be used for query optimization.

### 3.2 Query Plans

A query plan indicates the sequence of invocations of services and their conjunctive composition through joins. The specification of a query plan allows the execution of a query as a dataflow computation, from the user's input to the production of **k tuples**, where  $k$  is a parameter of the optimization. Every tuple includes contributions from the various service calls which are progressively composed according to the dataflow. Result tuples can be guaranteed to be the **top-k** tuples according to the ranking function, or instead be just **k good** tuples, emitted with an approximation of the total order expressed by the ranking function. Top-k tuples are generated by query plans which use top-k join methods, described in the next chapter. All other plans use the join methods described in Section 4, which do not guarantee top-k results, but are normally faster than top-k join methods.

We represent plans as directed acyclic graphs (DAGs) where:

- Every node represents either an atom in the conjunctive query (i.e., a service invocation), or a join, or a selection operation.
- Every arc indicates data flow and parameter passing from outputs of one service to inputs of another service.
- Atoms are partitioned into *exact* and *search* services. Exact services are distinguished between *proliferative* and *selective* and may be chunked, while search services are always proliferative and chunked. An exact service is selective if it produces in average less than one tuple per invocation (and therefore, in average, fewer output tuples than input tuples). An exact service that is not selective “per se” is said to be selective *in the context of a query* when the query includes a selection predicate over the output attributes of the service and the combined execution of the exact service call and of the selection produces fewer output tuples than input tuples.
- Joins are either performed as *pipe joins* or as *parallel joins*. Pipe joins occur when the query in the plan is made feasible through a strategy which induces an

---

<sup>3</sup> The case of opaque rankings can be dealt with by associating the position of tuples in the result with a new attribute and then translating the position into a score in the  $[0..1]$  interval.

I/O dependency between two services, whereas parallel joins occur when there is no such dependency. Parallel joins are represented by explicit nodes, marked with an indication of the join strategy to be employed, while pipe joins are represented simply by cascading two service invocations.

- Selection nodes express selection or join predicates which cannot be performed either by calling services or by using connection patterns.<sup>4</sup> Each predicate is independently evaluated on tuples representing intermediate or final query results, immediately after the service call that makes the selection or join predicates evaluable.
- Two explicit nodes represent the query input (i.e., the process of reading INPUT variables, mapping onto the arguments of services and joins, and starting query execution) and output (i.e., returning tuples to the query interface).

The graphical syntax for representing query plans is represented in Fig. 1.

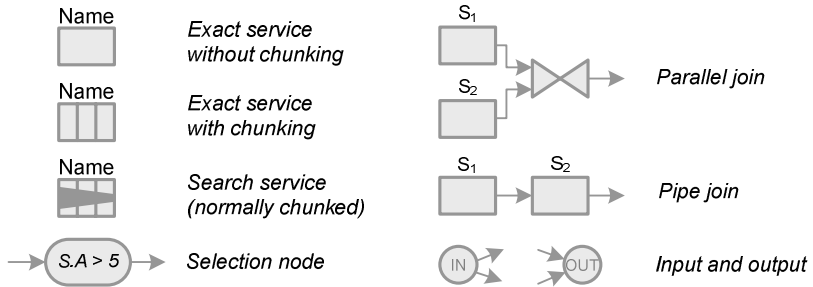


Fig. 1. Elements of query plans

We assume that services are independent of each other and that at each service call the values are uniformly distributed over the domains associated to their input and output fields. These assumptions allow us to obtain estimates for predicate selectivity and sizes of results returned by each service call. Cost models use estimates of the average result size of exact services and of chunk sizes.

An example of query plan representing the access to four services is shown in Fig. 2. The plan consists first in accessing two exact services named Conference and Weather. Conference is proliferative and produces 20 conferences on average, while Weather is selective in the context of the query, because extracted tuples are checked against the condition that the average temperature at the time of the conference must be above 26°C, and thus many of them can be discarded. Then, services describing flights to the conference city and hotels within that city are called, and their results are joined according to a given strategy, called merge-scan (MS), to be discussed later. Results of the join are transmitted to the user interface by the output node.

<sup>4</sup> These are expressed in the query and have the form:  $S_i.att_i \text{ op } const$  or  $S_i.att_i \text{ op } S_j.att_j$  where  $op$  is any comparison operation and attributes can be either single or multi-valued.

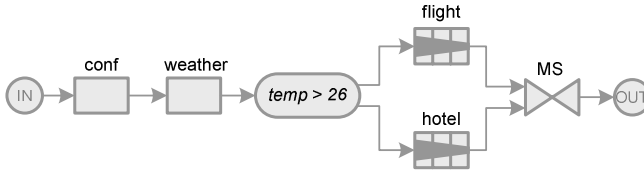


Fig. 2. Example of query plan

In our framework, we retrieve only the fraction of tuples of proliferative search services that are sufficient to obtain the first  $k$  tuples as query answers. We set  $k$  as optimization parameter so that the answered tuples should normally satisfy the user's needs (e.g.  $k=10$ ), but a plan execution can be continued, after an explicit user request, thereby producing more tuples. Therefore, a user can either be satisfied with the first  $k$  answers, or ask for more results of the same query, or change the choice of input keywords and resubmit the same query, or turn to a different query or Web activity. Moreover, if the strategy does not guarantee top- $k$  results, a query interface can be set so as to retrieve continuously tuples from the execution engine, without waiting for the extraction of  $k$  tuples. More details about user interaction are delayed to Chapter 13.

Also, for each node  $N$  in the plan we shall estimate the number of tuples in output, denoted as  $t_N^{\text{out}}$ . We assume that the user always injects one single input tuple in the plan, represented by the start node. For exact services,  $t_N^{\text{out}}$  is given by the product of  $t_N^{\text{in}}$  (the number of input tuples) with the service's average cardinality. For selection nodes,  $t_N^{\text{out}}$  is equal to  $t_N^{\text{in}}$  multiplied with the selectivity of the predicate. In both cases, numbers descend from the static properties of the query and can be computed from service interface statistics, under suitable independence and value distribution assumptions. Instead, if node  $n$  represents a join,  $t_N^{\text{out}}$  depends on the join selectivity and on the join method used; and if a node represents a search service,  $t_N^{\text{out}}$  is given by the product of the chunk size with the total number  $F_S$  of fetches determined by the plan, which may in turn depend on the input  $t_N^{\text{in}}$ . Therefore, the main decisions to be taken are join methods and access to search services. An annotated plan whose nodes are associated with  $t_N^{\text{in}}$ ,  $t_N^{\text{out}}$ , and  $F_S$  (if appropriate) is denoted as a fully instantiated query plans and can be associated with an execution cost. Fig. 3 shows an example of fully instantiated query plan obtained by annotating the plan of Fig. 2.

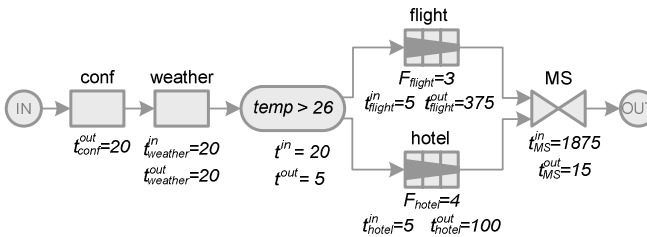


Fig. 3. Fully instantiated query plan, with annotations

## 4 Join Methods for Search Computing

In this section, we explore different join methods for search computing. We start by outlining the problem space and then continue to discuss three orthogonal characteristics of join methods, topology, invocation, and completion strategy. Finally, we discuss a number of concrete join methods that serve as a basis for upcoming chapters.

### 4.1 Problem Statement

Consider the join of two search services  $S_X$  and  $S_Y$ , and let  $R$  be the result of the join.  $R$  is a sequence of tuples  $r_k$  each obtained by joining two tuples  $x_i$ , produced by  $S_X$ , and  $y_j$ , produced by  $S_Y$ ;  $r_k$  is associated with a ranking function  $\rho^R_k$ , producing values within the  $[0..1]$  interval, obtained as the weighted sum of two scoring function  $\rho^X_i$  computed over  $t_i$  and  $\rho^Y_j$  computed over  $t_j$ . We can represent the chunks extracted from two services  $S_X$  and  $S_Y$  over the **axes of a Cartesian plan**, such that on each axis the ranking order of the chunks decreases from the origin down to the end of the list (see Fig. 4). Each point  $P$  in the plan represents a couple  $(x_i, y_j)$  which must be joined. If the join predicate holds, the point  $P$  belongs to the result. Services  $S_X$  and  $S_Y$  produce at each call a new chunk, named  $c_{X_i}$  and  $c_{Y_j}$  respectively, where  $c_{X_i}$  is a chunk returned by  $S_X$  in response to its  $i$ -th call and  $c_{Y_j}$  is a chunk returned by  $S_Y$  in response to its  $j$ -th call. The Cartesian plan is thus divided into rectangles with  $n_X \cdot n_Y$  points, where  $n_X$  and  $n_Y$  represent the chunk size of each service. We call *tile*  $t_{ij}$  the rectangular region that contains the points relative to chunks  $b_{X_i}$  and  $b_{Y_j}$ . Two tiles are said to be *adjacent* if they have one edge in common.

The plan is a model of the search space to be explored by a join operation. Each rectangular region of size  $m \cdot n$  represents the part of the search space that can be inspected after performing  $m$  request-responses to  $S_X$  and  $n$  request-responses to  $S_Y$ . Therefore, achieving extraction-optimality requires a suitable exploration strategy for such search space, which guides a “careful scan” of the result lists.

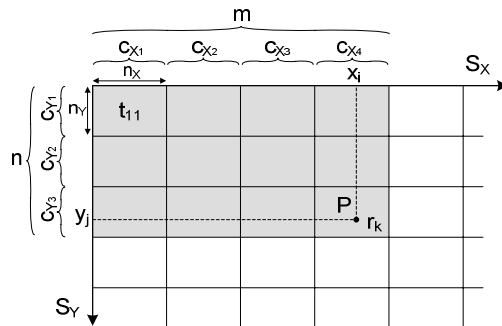


Fig. 4. Search Space for join operations

A join strategy is **optimal** if it produces, with the minimum cost,  $k$  tuples with the top- $k$  highest ranking. The cost of a strategy depends on the adoption of a specific cost model, whose factors include the cost of interacting with services and the cost of computing the join between service results. Cost-based optimal strategies for joining search services are the focus of the next chapter.

However, top- $k$  optimality is neither precise enough nor practically desired. First, rankings are sometimes approximate and the ranking function is rather arbitrary, thus reducing the practical relevance of producing top- $k$  results. Second, it may be inappropriate to produce results in strictly decreasing  $\rho^R$  values, because achieving such result normally requires halting the output production of result tuples until the system decides that all top- $k$  tuples are produced. So we introduce a revised notion of optimality, which only an approximate ranking of results.

If we assume that services return results in decreasing ranking order, we say that a join strategy is **extraction-optimal** if it produces elements  $r_k$  in decreasing order of the product of the two rankings  $\rho^X \cdot \rho^Y$  and with the minimum cost. Such notion extends from tuples to tiles by using the ranking of the first tuple of the tile as representative for the entire tile. Extraction-optimality enables the presentation of results which satisfy the join condition in the order in which they are computed, tile by tile. Therefore, the dataflow of results produced to the user is not “blocking” (by abusing of the terminology which is typical of query streams), and results can be presented to users while they are extracted from the search engines, typically arranged in chunks. The notion of extraction optimality can be further refined to be interpreted in *global* sense, i.e. relative to all the tiles in the search space, or in *local* sense, i.e. relative to the tiles already loaded in the search space and available to the join operation. If two tiles are adjacent, then the one with smaller index sum is extracted first by extraction-optimal methods.

Concerning the cost model, we consider the scenario in which the cost of join execution is dominated by request-response execution. We assume that once a chunk is retrieved as the effect of a request-response to services, then join requires simple main-memory comparison operations and can be neglected<sup>5</sup>. We further characterize the way in which ranking decreases (from top values close to 1 down to bottom values close to 0), by subdividing search services in the following two classes:

1. **Search Services with Step Scoring Function.** We assume that, by performing a limited number  $h$  of request-responses, most of the relevant entries will be retrieved, because the entry scores decrease with a deep step after  $h$  request-responses. We assume  $h$  to be a parameter associated with the service.
2. **Search Services with Progressive Scoring Function.** We assume that the scoring function decreases progressively, with no step. This case accommodates all regularly decreasing functions, e.g. linear or square value distributions.

---

<sup>5</sup> In previous work [3] we considered also the scenario where the cost of request-response execution is dominated by join execution. Such scenario considers more expensive “join” operations, e.g. the matching of terms which are extracted from a taxonomy or an ontology, where matching is expensive, e.g. because each element comparison requires itself a call to a semantic Web service.

Note that the unavailability of the ranking function does not affect our basic assumption that the search services return results in ranking order, but simply captures the situation in which this function is opaque. However, if the function is opaque, then classifying services and determining  $h$  in the former case is more difficult.

## 4.2 Topology

The first characteristic of a join method is its topology. When joining two search services, there are basically two possible ways of invoking the services. Either the services are invoked *sequentially* or *in parallel*. In the context of this book, the former case is referred to as a **pipe join**, while we will call the latter a **parallel join**.

### 4.2.1 Pipe Joins

Pipe joins use the fact that the access patterns of certain search services accept input parameters. In a sequence of services, the first service returns chunks of tuples that are passed down the sequence. A subset of the attributes of these tuples is the set of join attributes of a pipe join, whose values are passed, or “piped”, to another service that appears later in the sequence. These values are used as input values of the latter service’s calls, so as to produce a set of result tuples, obtained by composing the input tuple with the service call results. Note that in order to perform a pipe join, the two search services do not need to follow one another directly in the sequence of services. Also, multiple pipe joins can be performed within a sequence of search services. As shown in Fig. 1, pipe joins are not represented by any dedicated symbol in query plans. Rather they are just a sequence of service invocations that are chained by passing the output of one invocation as input to the next.

### 4.2.2 Parallel Joins

Parallel joins enable parallelizing the invocation of Web services and are fundamental operations of query plans, where they are represented by means of dedicated nodes, shaped as a join symbol. Binary parallel joins have been studied in [3].

## 4.3 Invocation Strategy

Apart from the topology, a join method is characterized by the order and frequency in which the services involved in a join are invoked. We refer to this property as the *invocation strategy*. The choice of invocation strategy depends on the distribution of the ranking of the results and the cost of service invocation. We consider two cases named nested-loop and merge-scan, for their analogy to well-known join methods. In addition, other specific invocation strategies can be arbitrarily defined.

### 4.3.1 Nested-Loop

The nested loop strategy is suitable when the results of one search engine, conventionally the first service, exhibits a clear “step” (as defined in Section 4.1). In such case, we assume that the ranking of that service suddenly drops from a high value to a very low value. The corresponding best exploration strategy of the search space reminds of the “nested-loop” method for relational joins. The exploration consists of extracting all the  $h$  chunks corresponding to the high ranking values of the

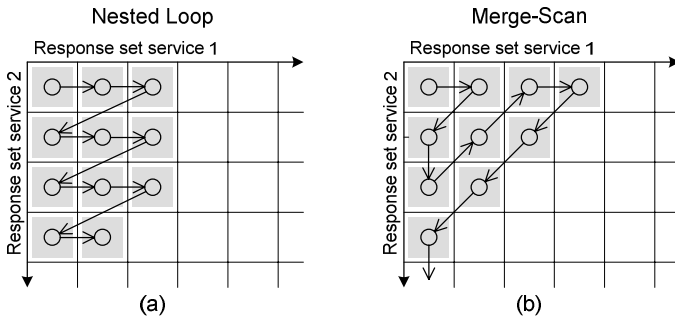


Fig. 5. Nested Loop (a) and Merge-Scan (b) strategies

“step” engine, and then extracting the chunks of the other service in ranking order, thereby producing join results. This strategy is represented in Fig. 5a.

### 4.3.2 Merge-Scan

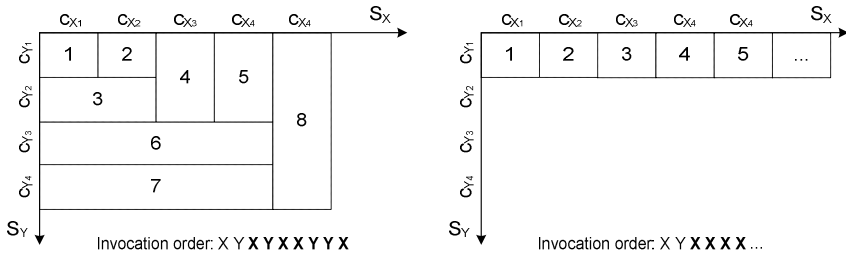
The merge-scan strategy is indicated in the absence of information about a clear “step” in the ranking of results. Then, one should assume that rankings decrease progressively. The corresponding best exploration strategy of the search space reminds of the “merge-scan” method for relational joins. The exploration consists in moving “diagonally” in the Cartesian plan, as shown in Fig. 5b, where the arrows indicate the order in which the tiles are chosen starting from the first tile. The method could evenly alternate service calls in the lack of better estimates of the score functions, or else it could use an **inter-service ratio**  $r$  between calls to services, and such ratio could be fixed (e.g.  $r=3/5$ ) or variable. Chapter 11 presents top- $k$  optimal join methods whose invocation strategy is merge-scan with variable inter-service ratios, based upon service costs. In Chapter 12 we show units for controlling the execution strategy, called clocks, whose function is to regulate service calls based upon the inter-service ratio.

## 4.4 Completion Strategy

Orthogonal to the invocation strategy that controls in which order and how often services are invoked, the *completion strategy* governs the order in which the tiles are considered by the join operation. Taken together, invocation and completion strategy thus control the exploration of the search space.

### 4.4.1 Rectangular

A rectangular strategy processes all the tiles as soon as the corresponding tuples are available. This completion strategy applies both to nested-loop and merge-scan. The rectangular strategy is locally extraction-optimal. With the nested loop method, if the step scoring function of the first service drops from 1 to 0 exactly in correspondence to the  $h$ -th chunk, then the method is globally extraction-optimal. A rectangular strategy matched to a particular sequence of requests is shown in Fig. 6.



**Fig. 6.** Examples of rectangular completion strategies

It should be noted that a strong asymmetry in the ranking of the two services may lead to a “long and thin” rectangular completion strategy, composed of the already explored tiles. This degenerates, in the worst case, to addressing all the calls to one service only (except for the first two calls, which are always alternated so as to have at least one tile for starting the exploration). This particular case, shown in Fig. 6, has the disadvantage that each I/O only adds one tile.

#### 4.4.2 Triangular

A triangular strategy processes all the tiles by moving “diagonally” in the Cartesian plan, as in the case of merge-scan, where a diagonal is expressed as ratio  $r=r_1/r_2$  between numbers of blocks in the search space. Thus, the method processes tiles  $t_{xy}$  such that the sum of indexes of two consecutive tiles extracted by the strategy cannot increase by more than one and that  $x r_2 + y r_1 < c$ , where  $c$  are constant values which are progressively increased by the method, starting with  $c= r_1 r_2$ . The triangular extraction strategy is locally extraction-optimal. When matched with the merge-join invocation strategy, it approximates an extraction-optimal strategy.

#### 4.5 Join Methods

This classification—topology, invocation and completion strategy—gives rise to eight possible methods for the join of two services. Note that not all combinations that would be theoretically possible also make sense in practice.

As a very simple example of a method that makes sense, Fig. 7 shows a rectangular completion applied to a merge scan in which the inter-service ratio is fixed to 1, resulting in the exploration of squares of increasing size. This method typically makes sense for parallel joins in which chunks are fetched alternatively from the two joined services. Pipe joins are better performed via nested loops with rectangular completion, which corresponds to retrieving the same number of fetches from the second service for each invocation originating from each tuple in output from the first service.

An example of method that makes little sense in practice is a rectangular completion applied to nested loop.



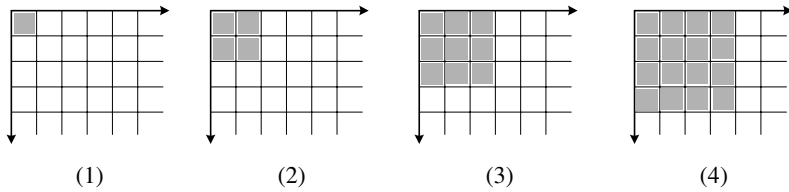


Fig. 7. Merge-scan, rectangular join strategy

## 5 Query Optimization for Search Computing

The optimization problem considered in this Section is: *given a query over a set of services, find the query plan that minimizes the expected execution cost according to a given cost metric in order to obtain the best  $k$  answers*. The process of generating an optimal plan starts from the conjunctive query over services (either service marts or service implementations) and ends with a fully instantiated invocation schedule.

### 5.1 Cost Metrics

A cost metric is a function that associates a cost to each query plan. We mainly consider the following two cost metrics:

**Execution time metric**, which measures the (expected) time elapsed from the query submission time to the production of the  $k$ -th answer. The time required for producing  $k$  tuples takes into account the number of invocations of each unit and the expected elapsed time for the execution of that unit in order to obtain a given number of results. The cost must account for the slowest path flowing tuples from the input to the output of the plan.

**Sum cost metric**, which computes the cost of a plan for producing  $k$  answers as the sum of the costs of each operator used in the plan. Examples of costs for a service invocation are the cost of computing joins or the cost charged by the service. A special case of the sum cost metric is the request-response cost metric, which consists of considering only the cost of service invocations required to execute the plan, omitting to consider operation execution costs. A further simplification is to assume that every service invocation has the same cost, and in such case the metric simply counts the number of calls. This metric is particularly relevant when the transfer of data over the network is the dominating cost factor.

There are other cost metrics of interest, though not considered in detail in the rest of the chapter:

**Bottleneck cost metric**, which gives the execution time of the slowest service in the plan, and is relevant in contexts of pipelined execution of continuous queries. This metric, extensively studied in [22], is suitable to contexts with homogeneous services (resembling a distributed DBMS) but it is not advised in our context, where search services rarely produce all their tuples and the execution is normally limited to reaching  $k$  answers.

**Time-to-screen cost metric**, which measures the time required to present the user with the first output tuple. This metric is suitable for settings in which the user expects a prompt interaction.

## 5.2 Branch and Bound Approach to Query Optimization

After characterizing services, query plans, and cost metrics, we now introduce our optimization method, summarized in Fig. 8. The method explores the combinatorial solution space of all possible translations of the conjunctive query into fully instantiated invocation schedules. The exploration is organized by means of an incremental construction of the query plans, which takes place in three phases, imposing a discipline in the order in which alternative plans are generated and considered.

The first phase is the **selection of specific access patterns and service interfaces** for each service  $s_i$  occurring in the conjunctive query, so that the resulting query is proven as feasible<sup>6</sup>. This phase is required when the query is formulated at the highest level of abstraction, over service marts, and includes the selection of the “best” service interfaces. In the context of this chapter, service interfaces are already selected by the query, but feasibility has to be proven. If no feasible plan can be generated for a given query, the translation fails. Otherwise, the chosen service interfaces are passed to the second phase to set up the query topology.

The second phase is the **selection of a query topology** for the given choice of service interfaces. This phase fixes the order of invocation of the services, as well as the data flow and the details of join operations. Indeed, it is worth reminding that even when all access patterns have been determined, there may still be several alternative DAGs compatible with the precedence constraints they enforce.

The third phase is the **choice of the number of fetches** to be performed over the chunked services. This phase allows to fully determine the execution schedule and the join strategies, and therefore to compute its cost according to a given metric.

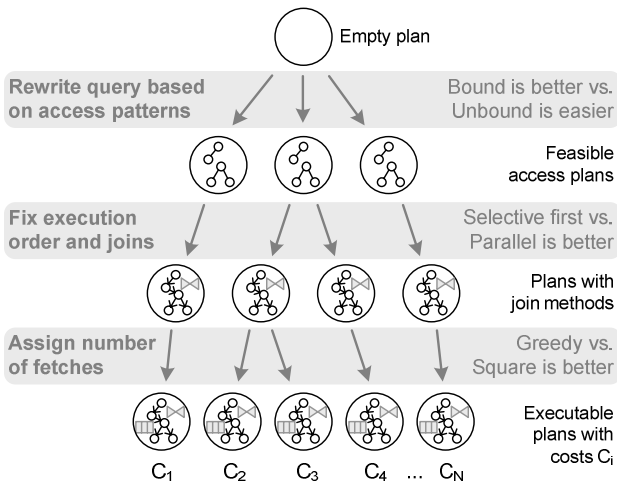


Fig. 8. Branch and bound

<sup>6</sup> Queries may also be formulated by means of syntax-aware user interfaces, such as the mashup environment described in [5]; in such cases, the query is guaranteed to be feasible by construction, as the tools do not allow users to compile unfeasible queries.

Each phase is combinatorial and the considered problem is hardly tractable by exact methods, even with queries involving few services. However, all considered cost metrics are monotonic, which allows for an exploration of the whole search space with a branch and bound strategy.

The incremental construction of query plans starts from an empty plan. Each choice in any of the three phases determines a subdivision of the search space into non-overlapping subsets, which is an ideal branching. Then, thanks to the mentioned monotonicity, each subset can be assigned a *lower bound* for the cost by calculating the cost on the partially constructed plan. To complete the *bounding* step, we can obtain an *upper bound* for a class of plans by fully constructing one plan in the class and calculating its cost. With this, we may apply the *pruning step*: if the lower bound for some class A is greater than the upper bound for some other class B, then A (and, implicitly, all solutions derivable from the elements of A) may be safely discarded from the search. In this way, the method converges to a local optimum, which under restrictive assumptions coincides with the global optimum.

This approach is traditionally used within database optimizers; e.g., the analogous phases in join optimization consist first in determining the join order, then the join method, then its parameterization according to the supported join execution procedures. Our problem has a similar combinatorial explosion, and we have evidence (thorough prototype implementations, e.g., in [4]) that the optimization can find reasonably good solutions in acceptable execution time.

In the following, for each of the three phases we (i) define the branching and bounding steps to be used for examining the solution space, and (ii) describe heuristics for choosing the branches so as to build efficient plans quickly. The search for the optimal plan can be stopped at any time, and it will nevertheless return a valid solution. If let run up to exhaustion of the search space, the returned plan is the optimal one, otherwise the returned plan is the one representing the current upper bound, whose “distance” from the optimal one depends on the effectiveness of heuristic choices and the running time of the algorithm.

### 5.3 Phase 1: Access Pattern Selection

Query plans are to be constructed taking feasibility into account. Initially, all atoms in the query are considered. At least one atom must be reachable based on available access patterns for that atom, or else the query is not feasible. Selected atoms are associated to a service interface, which is either chosen by the system or by the user within all service interfaces with that access pattern. The association of the first atom to an access pattern corresponds to a family of access plans, obtained by setting all the first atom’s attributes as bound, and therefore turning some other atoms as reachable. The process continues with branches driven by heuristics and bounds given by the cost of complete plans, unless some atom cannot be reached and the query is declared unfeasible. Lower bounds can be computed, e.g., by isolating the services that have fewer bound input attributes than some services in an already computed solution, and then by computing the cost associated to those services under the most favorable assumptions. The bound is effective if such cost exceeds the complete cost of the considered solution. The choice of the next atom can be done according to one of the following heuristics:

- **Bound is better:** a good heuristics for the choice of access patterns consists in preferring those with many input attributes. The intuition behind this heuristics is that the more attributes are bound to a given input, the smaller is the answer set, and therefore the service is faster in producing results, and less memory is required to cache the data.
- **Unbound is easier:** the shortcoming of the previous heuristics is that with many input attributes it is more difficult to find an assignment that makes the query feasible. Therefore, in contrast to the previous heuristics, an initialization with the minimum number of input attributes may make it easier to build a feasible solution.

#### 5.4 Phase 2: Selection of a Query Topology

The construction of all possible DAGs for a query plan can be done incrementally. It starts by placing after the initial node some node corresponding to a reachable service, and then by progressively adding nodes corresponding to services that are reachable by virtue of the user input variables and the services already included in the query. Nodes can be added in series or in parallel with respect to already included nodes, compatibly with the constraints enforced by I/O dependencies. Clearly, the space of constructible DAGs may grow very quickly, due to the exponential number of choices, multiplied at each step of the construction. Yet, the number of choices also depends on the degrees of freedom on the partial order induced by the access patterns. indeed, if the access patterns determine a total order, then there is only one possible DAG. The choice of the next node in the query plan can be done according to one of the following heuristics.

- **Selective first:** having long linear paths in the DAG, ordered by decreasing selectivity, wherever possible (ideally, one chain from input to output).
- **Parallel is better:** always making the choice that maximizes parallelism. Of course, this does not necessarily minimize the cost. Generally speaking, incrementing the parallelism plays in favor of those metrics that take time into account, while sequencing selective services plays in favor of metrics that minimize the overall number of invocations. In absence of access limitations, this gives the optimal solution, as proved in [22].

#### 5.5 Phase 3: Choice of the Number of Fetches

Whenever a query includes chunked services, say  $cs_1, \dots, cs_M$ , we need to provide an estimate of the number of chunks that will be fetched per input tuple at each  $cs_i$ . We call this numbers the *fetching factors* of the services, represented as a n-uple  $\langle F_1, \dots, F_M \rangle$ . Initially, all fetching factors are set to 1, which is the lowest admissible value for such parameters, as all services must contribute to the result. Clearly, if the n-tuple  $\langle 1, 1, \dots, 1 \rangle$  already determines  $h \geq k$  results, then it is also the optimal solution. otherwise, the fetching factors have to be incremented, until  $h \geq k$ . This can be done incrementally, according to one of the following heuristics.

- **Greedy:** at each iteration, the  $F_i$  to be incremented is the one that corresponds to the node in the plan with the *highest sensitivity* with respect to the increase in the number of tuples in the query result *per cost unit*. Computing such

sensitivity parameter is rather complex as it takes into account the query topology, as the increase of tuples in output in one node causes more tuples to be processed (and hence costs) in all the successors of that node.

- **Square is better:** at each iteration, each  $F_i$  is incremented by a value that is proportional to its chunk size. This implies that, in average, after query execution, all chunked services will have explored about the same number of tuples. The name of the heuristics originates from the fact that the fetching factors are incremented in such a way that the explored parts of the search space of all binary joins are kept square and of the same size.

## 5.6 Optimization Applied to the Running Example

We now briefly show how optimization can be applied to the query of the running example. The query is already formulated over service interfaces, whose adornments are as follows:

Theatre<sub>1</sub> (Name<sup>○</sup>, UAddress<sup>↓</sup>, UCity<sup>↓</sup>, UCountry<sup>↓</sup>, TAddress<sup>○</sup>, TCity<sup>○</sup>, TCountry<sup>○</sup>,  
TPhone<sup>○</sup>, Distance<sup>R</sup>, Movie.Title<sup>○</sup>, Movie.StartTimes<sup>○</sup>, Movie.Duration<sup>○</sup>)

Movie<sub>1</sub> (Title<sup>○</sup>, Director<sup>○</sup>, Score<sup>R</sup>, Year<sup>○</sup>, Genres.Genre<sup>↓</sup>, Language<sup>○</sup>,  
Openings.Country<sup>↓</sup>, Openings.Date<sup>↓</sup>, Actor.Name<sup>○</sup>)

Restaurant<sub>1</sub> (Name<sup>○</sup>, UAddress<sup>↓</sup>, UCity<sup>↓</sup>, UCountry<sup>↓</sup>, RAddress<sup>○</sup>, RCity<sup>○</sup>,  
RCountry<sup>○</sup>, Phone<sup>○</sup>, Url<sup>○</sup>, MapUrl<sup>○</sup>, Distance<sup>R</sup>, Rating<sup>R</sup>, Category.Name<sup>↓</sup>)

With this choice of access patterns the query is feasible. Referring back to the conjunctive formulation of Section 3.1, we note that all input attributes of Theatre and Movie are covered by INPUT variables, and the three input attributes of Restaurant are joined with the homonymous ones that are in output in Theatre. This test ends the first phase, identifying the I/O dependency that holds from Theatre to Restaurant. As for the second phase, four topologies are to be considered, shown in Fig. 9.

In all configurations Theatre precedes Restaurant, so as to implement with a pipe join the corresponding I/O dependency. Among the alternatives, we choose to continue our example with (d), which also contains a parallel join (as it would be chosen by the heuristics “parallel is better”).

We then set  $K = 10$  as the number of desired output combinations, and estimate the values of some basic parameters. Selectivity of the join predicates and chunk sizes are essential to calculate  $t^{\text{in}}$  and  $t^{\text{out}}$  for all the nodes in the plan. We estimate the selectivity of Shows() and DinnerPlace() as 2% and 40% respectively – namely, the probability that a given movie is being shown in a given theatre and the probability that a given theatre is placed close to a good restaurant. The value of  $K$  can be “back-propagated” through the nodes of the plan, as follows:  $K = 10$  implies  $t_{\text{Restaurant}}^{\text{out}} = 10$ .



Fig. 9. Alternative topologies for the running example

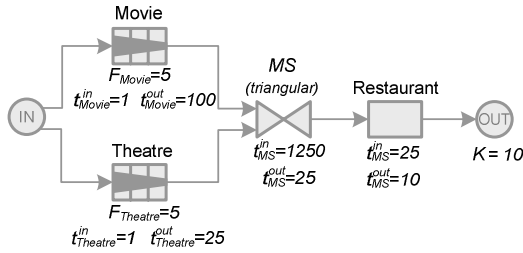


Fig. 10. Fully instantiated query plan for the running example

We choose to only keep and include in the result the first (and presumably best!) restaurant found for each location, therefore  $t_{\text{Restaurant}}^{\text{in}} = 25$ , by virtue of the selectivity of the pipe join. This in turn implies  $t_{\text{MS}}^{\text{out}} = 25$ , and therefore that the parallel join has to process 1250 candidate combinations overall. Here the space of possible solutions opens up quite widely, as different numbers of invocations and numbers of fetches per invocation can be assigned in order for 1250 combinations to be generated. Assuming that we restrict to the first 100 movies, corresponding to 5 fetches of chunks of 20 movies and to the first 25 theatres in order of distance from the user’s address, corresponding to 5 chunks of size 5, we can consider the fully instantiated query plan obtained by annotating the plan of Fig. 10. Note that multiplying  $t_{\text{Movie}}^{\text{out}} = 100$  by  $t_{\text{Theatre}}^{\text{out}} = 25$  we obtain 2500, but choosing a triangular completion strategy assures that only the half of the “most promising” combinations (either close theatre or very good movies) are considered, thus obtaining  $t_{\text{MS}}^{\text{out}} = 1250$ .

We have only considered one possible instantiation of the query. It would be the branch and bound’s responsibility to apply the cost metric to this “initial” plan, consider it as an upper bound, and explore the search space for less costly solutions.

## 6 Conclusions

This chapter has presented a formal model for the optimization and the execution of multi-domain queries over services. We have turned a high-level formulation of queries over services into executable query plans, describing the invocations of Web services and the composition of their inputs and outputs, and then we have presented a method for optimizing the selection of access plans according to cost metrics. Access plans include as its main ingredient the join between service results, and in this chapter we have reviewed efficient methods for join execution, which however do not guarantee that results are the top-k according to a global ranking. This chapter has therefore created the premises for the next two chapters, where we describe methods for rank aggregation which guarantee top-k results, and an engine for query plan execution.

## References

1. Altinel, M., Brown, P., Cline, S., Kartha, R., Louie, E., Markl, V., Mau, L., Ng, Y.-H., Simmen, D., Singh, A.: Damia - A Data Mashup Fabric for Intranet Applications. In: VLDB 2007, pp. 1370–1373 (2007)

2. Bianchini, D., De Antonellis, V., Pernici, B., Plebani, P.: Ontology-based Methodology for e-Service Discovery. *Inf. Syst.* 31(4-5), 361–380 (2006)
3. Braga, D., Campi, A., Ceri, S., Raffio, A.: Joining the Results of Heterogeneous Search Engines. *Inf. Syst.* 33(7-8), 658–680 (2008)
4. Braga, D., Ceri, S., Daniel, F., Martinenghi, D.: Optimization of Multi-Domain Queries on the Web. *PVLDB* 1(1), 562–573 (2008)
5. Braga, D., Ceri, S., Daniel, F., Martinenghi, D.: Mashing Up Search Services. *Internet Computing* 12(5), 16–23 (2008)
6. Calì, A., Martinenghi, D.: Querying Data under Access Limitations. In: *ICDE 2008*, Cancún, Mexico, pp. 50–59 (2008)
7. Chamberlin, D.D., Astrahan, M.M., King, W.F., Lorie, R.A., Mehl, J.W., Price, T.G., Schkolnick, M., Selinger, P.G., Slutz, D.R., Wade, B.W., Yost, R.A.: Support for Repetitive Transactions and Ad Hoc Queries in System R. *ACM-TODS* 6(1), 70–94 (1981)
8. Confalonieri, R., Domingue, J., Motta, E.: Orchestration of Semantic Web Services in IRS-III. In: *AKT-SWS 2004*. The Open University, Milton Keynes (2004)
9. Daniel, F., Pernici, B.: Insights into Web Service Orchestration and Choreography. *International Journal of E-Business Research* 2(1), 58–77 (2006)
10. Deutsch, A., Ludäscher, B., Nash, A.: Rewriting Queries using Views with Access Patterns under Integrity Constraints. *Theoretical Computer Science* 371(3), 200–226 (2007)
11. DeWitt, D.J., Ghandeharizadeh, S., Schneider, D.A., Bricker, A., Hsiao, H.-I., Rasmussen, R.: The Gamma Database Machine Project. *IEEE Trans. on Knowledge and Data Engineering* 2(1), 44–62 (1990)
12. Florescu, D., Levy, A.Y., Manolescu, I., Suciu, D.: Query Optimization in the presence of Limited Access Patterns. In: *SIGMOD 1999*, Philadelphia, Pennsylvania, USA, pp. 311–322 (1999)
13. Ioannidis, Y.E., Kang, Y.: Randomized Algorithms for Optimizing Large Join Queries. *SIGMOD Rec.* 19(2), 312–321 (1990)
14. Ives, Z.G., Halevy, A.Y., Weld, D.S.: Adapting to Source Properties in Processing Data Integration Queries. In: *SIGMOD 2004*, Paris, France, pp. 395–406 (2004)
15. Kossmann, D., Stocker, K.: Iterative Dynamic Programming: a New Class of Query Optimization Algorithms. *ACM-TODS* 25(1), 43–82 (2000)
16. Li, C., Chang, E.: Answering Queries with Useful Bindings. *ACM-TODS* 26(3), 313–343 (2001)
17. Lohman, G.M.: Grammar-like Functional Rules for Representing Query Optimization Alternatives. In: *SIGMOD 1988*, Chicago, Illinois, USA, pp. 18–27 (1988)
18. Millstein, T.D., Levy, A.Y., Friedman, M.: Query Containment for Data Integration Systems. In: *PODS 2000*, Dallas, Texas, USA, pp. 67–75 (2000)
19. OASIS: Web Services Business Process Execution Language. Technical report (2007), <http://www.oasis-open.org/committees/wsbpel/>
20. Rajaraman, A., Sagiv, Y., Ullman, J.D.: Answering Queries using Templates with Binding Patterns. In: *PODS 1995*, San José, California, USA, pp. 105–112 (1995)
21. Seshadri, P., Hellerstein, J.M., Pirahesh, H., Cliff Leung, T.Y., Ramakrishnan, R., Srivastava, D., Stuckey, P.J., Sudarshan, S.: Cost-based Optimization for Magic: Algebra and Implementation. *SIGMOD Rec.* 25(2), 435–446 (1996)
22. Srivastava, U., Munagala, K., Widom, J., Motwani, R.: Query Optimization over Web Services. In: *VLDB 2006*, Seoul, Korea, pp. 355–366 (2006)

23. Özsu, M.T., Valduriez, P.: Principles of Distributed Database Systems. Prentice-Hall, Inc., Upper Saddle River (1991)
24. Tao, Y., Hristidis, V., Papadias, D., Papakonstantinou, Y.: Branch-and-Bound Processing of Ranked Queries. *Inf. Syst.* 32(3), 424–445 (2007)
25. Tatemura, J., Sawires, A., Po, O., Chen, S., Candan, K.S., Agrawal, D., Goveas, M.: Mashup Feeds: Continuous Queries over Web Services. In: SIGMOD 2007, New York, NY, USA, pp. 1128–1130 (2007)
26. W3C: Web Services Choreography Description Language, Version 1.0. W3C Candidate Recommendation (2005), <http://www.w3.org/TR/ws-cdl-10/>
27. W3C: Web Service Choreography Interface (WSCI), Version 1.0. W3C Note (2002), <http://www.w3.org/TR/wsci/>
28. WSMO: Web Service Modeling Ontology, <http://www.wsmo.org>
29. Yang, G., Kifer, M., Chaudhri, V.K.: Efficiently Ordering Subgoals with Access Constraints. In: PODS 2006, Chicago, Illinois, USA, pp. 183–192 (2006)
30. Yu, C.T., Meng, W.: Principles of Database Query Processing for Advanced Applications. Morgan Kaufmann, San Francisco (2005)



# Chapter 11:

## Rank-Join Algorithms for Search Computing

Ihab F. Ilyas<sup>1</sup>, Davide Martinenghi<sup>2</sup>, and Marco Tagliasacchi<sup>2</sup>

<sup>1</sup> University of Waterloo, Canada

<sup>2</sup> Politecnico di Milano

ilyas@uwaterloo.ca, martinen@elet.polimi.it,  
tagliasa@elet.polimi.it

**Abstract.** Joins represent the basic functional operations of complex query plans in a Search Computing system, as discussed in the previous chapter. In this chapter we provide further insight on this matter, by focusing on algorithms that deal with joining ranked results produced by search services. We cast this problem as a generalization of the traditional rank aggregation problem, i.e., combining several ranked lists of objects to produce a single consensus ranking. Rank-join algorithms, also called top-k join algorithms, aim at determining the best overall results without accessing all the objects. The rank-join problem has been dealt with in the literature by extending rank aggregation algorithms to the case of join in the setting of relational databases. However, previous approaches to top-k queries did not consider some of the distinctive features of search engines on the Web. Indeed, as pointed out in the previous chapter, joins in this context differ from the traditional relational setting for a number of aspects: services can be accessed according to limited patterns, i.e. some inputs need to be provided; accessing services is costly, since they are typically remote; the output is returned in pages of results and typically according to some ranking criterion; multiple search services can be used to answer the same query; users can interact with the system in order to refine their search criteria. This chapter analyzes the challenges that need to be tackled in the design of rank-join algorithms within the context of Search Computing.

**Keywords:** rank-join, top-k, query optimization.

## 1 Introduction

Information systems of different types use various techniques to rank query answers. In many application domains, end-users are more interested in the most important (top-k) query answers in the potentially huge answer space. Different emerging applications warrant efficient support for top-k queries. For instance, in the context of the Web, the effectiveness and efficiency of meta-search engines are highly related to efficient methods for rank aggregation. The latter is the problem of combining several ranked lists of items in a robust way to produce a single consensus ranking of the items. Most of these applications execute queries that involve joining and aggregating multiple inputs to provide the top-k results.

We focus here on a specific class of top-k processing techniques, known as *rank join* algorithms, that retrieve the top-k combinations among a set resulting from joining multiple sources. Such techniques are crucial for answering multi-domain queries. This requires extracting and combining the answers of domain-specific search systems. Then, a global ranking needs to be established for each combined answer by means of an aggregation function, so as to present to the user the answers having the top combined scores.

The data sources we consider in this chapter, which are typical of search computing scenarios, may be asked to output data tuples sorted by score, where the score is an explicit field of the tuples. For example, a data source can consist of the output of a Web service or a wrapped Web site. The ranked lists may contain a large number of objects, usually presented in pages, and accessing such pages typically comes at a non-negligible cost. Moreover, objects can be accessed according to various methods, that can be broadly classified as sorted, producing a possibly long ranked list of objects (whose tail is typically uninteresting), or random, producing a narrower set of objects, normally not ranked, which satisfy a selection condition over the attributes.

This chapter is organized as follows. In Section 2, we introduce the rank-join problem and identify the specific needs and challenges it poses in the context of search computing. In Section 3, we review existing methods and algorithms, while in Section 4 we outline optimization opportunities for rank-join in search computing.

## 2 New Rank-Join Challenges in Search Computing

Answering complex queries such as those given in Chapter 1 requires combining information from different, heterogeneous search services. Typically, these sorts of services produce results ranked by score, which need to be joined in order to form combinations. Each combination is given a score, which is computed by aggregating the scores of the data items that have been used to form the combination. In this context, users are interested in exploring only a subset of the result, i.e. only the top combinations ordered by aggregated score. A naïve, yet inefficient, solution consists of: 1) retrieving all the data items from the data sources involved; 2) joining the results to form combinations; 3) computing their scores; and, finally, 4) sorting the combinations based on their scores. The inefficiency of such approach stems from the fact that most of the fetched data are not used to produce the result presented to the user. A similar problem has been addressed in the past literature in the context of database systems, where a family of solutions, known under the name of rank-join (or top-k join) algorithms, has been thoroughly explored. The underlying idea of rank-join algorithms, further detailed in Section 3, is to leverage the fact that input data sources, i.e. relational tables, are already sorted by score. Therefore, an early-out strategy can be defined, in such a way to stop fetching tuples as soon as it is possible to guarantee that the top-k combinations can already be formed. The goal of traditional rank-join algorithms is then to minimize the *I/O cost* with respect to the naïve join-then-sort approach.

While rank-join algorithms can be relevant in the context of search computing, there are several peculiarities that need to be addressed when data sources are search

services rather than relational tables. In the following we provide an overview of the distinct aspects that characterize the domain of search computing, which motivate the need for revisiting off-the-shelf rank-join algorithms.

**Access Patterns.** Unlike data in the relational setting, search services typically expose a limited number of access interfaces, in which certain fields must be mandatorily filled in, in order to obtain a result. These fields may be the input fields of a form on a data-intensive Web site or the input parameters of a Web Service invocation. In order to model such access limitations, we assume that each service is characterized by a given number of combinations of input and output parameters, called *access patterns*, corresponding to the different ways in which it can be invoked. Note that availability of access patterns is out of the control of the search computing system, which must therefore always determine an access strategy that complies with their requirements.

**Access Costs.** In a conventional relational setting, rank-join algorithms operate on local data that reside in a centralized database system. Conversely, we are concerned in joining the results of search services, which typically operate on remote servers. Fetch operations are costly and shall be explicitly taken into account by rank-join algorithms, i.e., they must become cost-aware. Moreover, access costs might depend on the specific services to be invoked as well as on the available access methods. In the past literature on rank-join, two access methods prevail: *sorted access* and *random access*.

Sorted access returns tuples sorted by score and is typically available for all search services, with the notable difference that each new invocation produces a page of data items instead of a single tuple. In some cases, there might be only one available ranking criterion for sorted access to a search service, while in other cases there might be several possibilities; for example, a search service may allow retrieving movies sorted either by user rating or by box office income, possibly with different access costs depending on the ranking criterion. Note that the query formulation might require sorted access to the service with a ranking criterion that is not available. For example, consider a query that wants to retrieve movies sorted by user rating, but the selected service can only be invoked in such a way that results are returned sorted by box office income. In such a case, if the correlation between stars and prices can be somehow modeled, rank-join algorithms can be adapted to answer such queries as well.

Random access returns tuples that refer to a given object, e.g. all the movie theatres in a given district of the city of Paris, and allows an earlier termination of rank-join algorithms when it is available, thus minimizing the number of I/O operations. In a relational setting, random access can be provided by building an index on top of one of the attributes of a table. This is not a viable option in the context of search computing. However, when a search service, say  $s_1$ , only provides sorted access, it is to some extent possible to obtain random access by invoking another search service, say  $s_2$ , returning data items of the same kind, although  $s_2$  might be characterized by a different access cost and contain only a subset (or a superset) of the data items of  $s_1$ . Moreover, random accesses in the search computing context, do not necessarily return all the data items referring to a given object, but, instead, only a subset might be retrieved, sometimes organized in pages and sorted by score.

**Redundancy of Data Sources.** In several domains, there might be different search services which can be potentially invoked to answer identical or similar queries. Consider, for example, the plethora of partially overlapping services that search for movies. Such a redundant availability of data sources comes at no additional cost in the context of search computing. If properly managed, it can be exploited as an asset in two ways: improving the system response time and the robustness to service failures or time-varying access costs. With respect to the first goal, parallel invocation of two or more services can be put in place in a rather straightforward way. Conversely, properly combining their results can be far from trivial due to the different underlying data populations, naming conventions, etc. Instead, with respect to the second goal, robustness can be achieved by adaptively switching to a different search service. At the same time, it is possible to re-use the results of the computations already performed, by carefully handling potential duplicates that might arise. In addition, the availability of multiple search services, each characterized by its access cost, might provide random access when no single service is able to do so. For example, different country-specific search services can be used to find movie theatres in a given European city, when there is none of them that cover exhaustively the whole territory.

**Users in the Loop.** The queries submitted by users of a search computing system can be dynamically shaped in order to help satisfying the user's information needs. The liquid queries paradigm further explored in Chapter 13 envisions a set of operations that can be performed at the client side. Some of these operations do not require interaction with the remote search services, since they only affect the visualization of data already available at the client side. Others, instead, require fetching additional data from remote services. For example, the user might want to dynamically adjust the aggregation function. In a weighted sum, this is accomplished by changing the weights assigned to the different search services. In order to preserve the guarantee of displaying the top results, further data might need to be fetched. If a statistical model describing the user interaction with the weights can be provided, then rank-join algorithms can be adapted to pre-fetch the data items that are more likely to be used and store them in a cache at the client side. Additional examples of non-trivial user interactions involve asking for results from other search services, either covering the same objects or complementing the information about already retrieved objects. In both cases re-use of already available information is mandatory to enable fast response times as required by the liquid query paradigm.

The visualization of the results of such complex queries might also affect the underlying rank-join algorithms. Consider a query that retrieves combinations of movies and movie theatres in the same district of the city of Paris, sorted by the combined user ratings. To some extent, the user might prefer to see only the first few combinations for each district, perhaps grouped by district, where the ordering among groups might depend on the leading (or average) combination score. The traditional rank-join problem needs to be revisited, since the goal has changed from retrieving top-k combinations to top-k groups. A related problem is that of result diversification addressed, e.g., in [17, 18, 19].

### 3 Rank-Join Algorithms: State of the Art

In this section, we review the main algorithms that were developed for the rank join problem. We also discuss related top-k processing techniques that inspired many of the rank join algorithms. In the following, we introduce a taxonomy to classify top-k techniques based on two design dimensions.

- *Data Access Methods*: Top-k processing techniques are classified according to the data access methods they assume to be available in the underlying data sources. For example, some techniques assume the availability of random access, while others are restricted to only sorted access.
- *Implementation Level*: Top-k processing techniques are classified according to their level of integration with database systems. For example, some techniques are implemented in an application layer on top of the database system, while others are implemented as query operators.

In the following sections, we discuss the design dimensions in details.

#### 3.1 Data Access

Many top-k processing techniques involve accessing multiple data sources with different valuations of the underlying data objects. The manner in which these sources are accessed largely affects the design of the underlying top-k processing techniques. For example, ranked lists could be scanned sequentially in score order. We refer to this access method as *sorted access*. On the other hand, the score of some object might be required directly without traversing the objects with higher/smaller scores. We refer to this access method as *random access*. Random access could be provided through index lookup operations if an index is built on object keys.

We classify top-k processing techniques, based on the assumptions they make about available data access methods in the underlying data sources, as follows:

- *Both Sorted and Random Access*: In this category, top-k processing techniques assume the availability of both sorted and random access in all the underlying data sources.
- *No Random Access*: In this category, top-k processing techniques assume that data sources provide only sorted access to objects based on their scores.
- *Sorted Access with Controlled Random Probes*: In this category, top-k processing techniques assume the availability of at least one sorted access source. Random accesses are used in a controlled manner to reveal the overall scores of candidate answers.

Next, we discuss the three categories.

##### 3.1.1 Both Sorted and Random Access

Top-k processing techniques in this category assume data sources that support both access methods, sorted and random. The *Threshold Algorithm (TA)* and *Combined Algorithm (CA)* [3] belong to this category.

Algorithm 1 describes the details of TA. The algorithm scans multiple lists, representing different rankings of the same set of objects. An upper bound  $T$  is

maintained for the overall score of unseen objects. The upper bound is computed by applying the scoring function to the partial scores of the last seen objects in different lists. Each newly seen object in one of the lists is looked up in all other lists, and its scores are aggregated using the scoring function to obtain the overall score. All objects with total scores that are greater than or equal to  $T$  can be reported. The algorithm terminates after returning the  $k^{\text{th}}$  output.

**ALGORITHM 1: TA – (Threshold Algorithm) [3]**

1. Do sorted access in parallel to each of the  $m$  sorted lists  $L_i$ . As a new object  $o$  is seen under sorted access in some list, do random access to the other lists to find  $p_i(o)$  in every other list  $L_i$ . Compute the score  $F(o) = F(p_1, \dots, p_m)$  of object  $o$ . If this score is among the  $k$  highest scores seen so far, then remember object  $o$  and its score  $F(o)$  (ties are broken arbitrarily, so that only  $k$  objects and their scores are remembered at any time).
2. For each list  $L_i$ , let  $\bar{p}_i$  be the score of the last object seen under sorted access. Define the *threshold value*  $T$  to be  $F(\bar{p}_1, \dots, \bar{p}_m)$ . As soon as at least  $k$  objects have been seen with scores at least equal to  $T$ , halt.
3. Let  $A_k$  be a set containing the  $k$  seen objects with the highest scores. The output is the sorted set  $\{(o, F(o)) \mid o \in A_k\}$ .

While TA assumes that the costs of different access methods are the same, the CA algorithm [3] alternatively assumes that the costs of different access methods are different. The CA algorithm defines a ratio between the costs of the two access methods to control the number of random accesses, since they usually have higher costs than sorted accesses. The CA algorithm periodically performs random accesses to collect unknown partial scores for objects with the highest score lower bounds (ties are broken using score upper bounds). A score lower bound is computed by applying the scoring function to object's known partial scores, and the *worst* possible unknown partial scores. On the other hand, a score upper bound is computed by applying the scoring function to object's known partial scores, and the *best* possible unknown partial scores. One random access is performed periodically every  $\Delta$  sorted accesses, where  $\Delta$  is the floor of the ratio between random access cost and sorted access cost.

### 3.1.2 No Random Access

The techniques we discuss in this section assume random access is not supported by the underlying sources. The Rank-Join algorithm [6] and the  $J^*$  algorithm [11] are examples of the techniques that belong to this category.

The Rank-Join algorithm [6] integrates the joining and ranking tasks in one efficient operator. Algorithm 2.2 describes the main Rank-Join procedure. The Rank-Join algorithm scans input lists (the joined relations) in the order of their scoring

predicates. Join results are discovered incrementally as the algorithm moves down the ranked input relations. For each join result  $j$ , the algorithm computes a score for  $j$  using a score aggregation function  $F$ . The algorithm maintains a threshold  $T$  bounding the scores of join results that are not discovered yet. The top- $k$  join results are obtained when the minimum score of the  $k$  join results with the maximum  $F()$  values is not below the threshold  $T$ .

**ALGORITHM 2: Rank join [2]**

1. Retrieve tuples from input relations in descending order of their individual scores  $p_i$ 's. For each new retrieved tuple  $t$ :
  - a. Generate new valid join combinations between  $t$  and seen tuples in other relations.
  - b. For each resulting join combination  $j$ , compute  $F(j)$ .
  - c. Let  $p_i^{(max)}$  be the top score in relation  $i$ , i.e., the score of the first tuple retrieved from relation  $i$ . Let  $\bar{p}_i$  be the last seen score in relation  $i$ . Let  $T$  be the maximum of the following  $m$  values:  

$$F(\bar{p}_1, p_2^{max}, \dots, p_m^{max}), F(p_1^{max}, \bar{p}_2, \dots, p_m^{max}), \dots, F(p_1^{max}, p_2^{max}, \dots, \bar{p}_m).$$
  - d. Let  $A_k$  be a set of  $k$  join results with the maximum  $F()$  values, and  $M_k$  be the lowest score in  $A_k$ . Halt when  $M_k \geq T$ .
2. Report the join results in  $A_k$  ordered on their  $F()$  values.

A two-way hash join implementation of the Rank-Join algorithm, called *Hash Rank Join Operator (HRJN)*, is introduced in [6]. HRJN is based on symmetrical hash join. The operator maintains a hash table for each relation involved in the join process, and a priority queue to buffer the join results in the order of their scores. The hash tables hold input tuples seen so far and are used to compute the valid join results. The HRJN operator implements the traditional iterator interface of query operators, which include two methods: *Open* and *GetNext*. The *Open* method is responsible for initializing the necessary data structure; the priority queue  $Q$ , and the left and right hash tables. It also sets  $T$ , the score upper bound of unseen join results, to the maximum possible value.

The *GetNext* method remembers the two top scores,  $p_1^{max}$  and  $p_2^{max}$ , and the last seen scores,  $\bar{p}_1$  and  $\bar{p}_2$  of its left and right inputs. At any time during query execution, the threshold  $T$  is computed as the maximum of  $F(p_1^{max}, \bar{p}_2)$  and  $F(\bar{p}_1, p_2^{max})$ . At each step, the algorithm reads tuples from either the left or right inputs, and probes the hash table of the other input to generate join results. The algorithm decides which input to poll at each step, which gives flexibility to optimize the operator for fast generation of join results based on the joined data. A simplistic strategy is accessing the inputs in a round-robin fashion. A join result is reported if its score is not below the scores of all discovered join results, and the threshold  $T$ .

In [12], an enhancement of HRJN algorithm is provided where a different bounding scheme is used to compute the threshold  $T$ . This is achieved by computing a

*feasible region* in which unseen objects may exist. Feasible region is computed based on the objects seen so far, and knowing the possible range of score predicates. The authors proved that the feasible region is tight, and thus the rank join algorithm that exploits such bounding scheme is instance optimal.

Another example of *no random access* top-k algorithms is the  $J^*$  algorithm [11]. The idea is to maintain a priority queue of partial and complete join combinations, ordered on the upper bounds of their total scores. At each step, the algorithm tries to complete the join combination at queue top by selecting the next input stream to join with the partial join result, and retrieving the next object from that stream. The algorithm reports the next top join result as soon as the join result at queue top includes an object from each ranked input.

### 3.1.3 Sorted Access with Controlled Random Probes

Top-k processing methods in this category assume that at least one source provides sorted access, while random accesses are performed only when needed. The Upper and Pick algorithms [2, 10] are examples of these methods.

Upper and Pick assume that each source can provide a sorted and/or random access to its ranked input, and that at least one source supports sorted access. The main purpose of having at least one sorted-access source is to obtain an initial set of candidate objects. In the Upper algorithm, candidate objects are retrieved first from sorted sources, and inserted into a priority queue based on their score upper bounds. The upper bound of unseen objects is updated when new objects are retrieved from sorted sources. An object is reported and removed from the queue when its exact score is higher than the score upper bound of unseen objects. The algorithm repeatedly selects the best source to probe next to obtain additional information for candidate objects. The selection is performed by a function, named *SelectBestSource*. This function could have several implementations. For example, the source to be probed next can be the one which contributes the most in decreasing the uncertainty about the top candidates.

In the Pick algorithm, the next object to be probed is selected so that it minimizes a distance function the most. Such distance function represents the sum of the differences between the upper and lower bounds of all objects. The source to be probed next is selected at random from all sources that need to be probed to complete the score of the selected object.

## 3.2 Implementation Level

Integrating top-k processing with database systems is addressed in different ways by current techniques. One approach is to embed top-k processing in an outer layer on-top of the database engine. This approach allows for easy extensibility of top-k techniques, since they are decoupled from query engines. The capabilities of database engines (e.g., storage, indexing and query processing) are leveraged to allow for efficient top-k processing.

Another approach is to modify the core of query engines to recognize the ranking requirements of top-k queries during query planning and execution. This approach has a direct impact on query processing and optimization. Specifically, query operators are modified to be rank-aware. For example, a join operator is required to produce



ranked join results to support pipelining top-k query answers. In this section, we discuss top-k processing methods based on the two design choices.

### 3.2.1 Application Level

Top-k query processing techniques that are implemented at the application level provide a ranked retrieval of database objects, without major modification to the underlying database system. We classify application level top-k techniques into Filter-Restart methods and Indexes/Materialized Views methods.

Filter-Restart techniques formulate top-k queries as *range selection queries* to limit the number of retrieved objects. That is, a top-k query that ranks objects based on a scoring function  $F$ , defined on a set of scoring predicates  $p_1, \dots, p_m$ , is formulated as a range query of the form “find objects with  $p_1 > T_1$ , and,  $\dots$ ,  $p_m > T_m$ ”, where  $T_i$  is an estimated cutoff threshold for predicate  $p_i$ . Using a range query aims at limiting the retrieved set of objects to the necessary objects to answer the top-k query. Incorrect estimation of cutoff threshold yields one of two possibilities: (1) if the cutoff is over-estimated, the retrieved objects may not be sufficient to answer the top-k query and the range query has to be restarted with looser thresholds, or (2) if the cutoff is under-estimated, the number of retrieved objects will be more than necessary to answer the top-k query.

One proposed method to estimate the cutoff threshold is using the available statistics such as histograms [1], where the scoring function is taken as the distance between database objects and a given query point  $q$ . Multidimensional histograms on objects' attributes (dimensions) are used to identify the cutoff distance from  $q$  to the potential top-k set. To find the optimal search distance, query workload is used as a training set to determine the number of returned objects for different search predicates.

Another group of *Application Level* top-k processing techniques use specialized indexes and materialized views to improve the query response time at the expense of additional storage space.

One example of specialized top-k indexes is *Ranked Join Indices* [13]. Ranked Join Indices is a top-k index structure, based on the observation that the projection of a vector representing a tuple  $t$  on the normalized scoring function vector  $\mathbf{w}$  reveals  $t$ 's rank based on  $\mathbf{w}$ . This observation applies to any scoring function that is defined as a linear combination of the scoring predicates. For example, consider a scoring function  $F = w_1 p_1 + w_2 p_2$ , where  $p_1$  and  $p_2$  are scoring predicates, and  $w_1$  and  $w_2$  are their corresponding weights. In this case, we have  $\mathbf{w} = [w_1, w_2]$ . Without loss of generality, assume that  $\|\mathbf{w}\| = 1$ . We can obtain the score of  $\mathbf{t} = [p_1, p_2]$  by computing the length of its projection on  $\mathbf{w}$ , which is equivalent to the inner product between  $\mathbf{w}$  and  $\mathbf{t}$ , i.e.  $w_1 p_1 + w_2 p_2$ . By changing the values of  $w_1$  and  $w_2$ , we can sweep the space using a vector of increasing angle to represent any possible linear scoring function. The tuple scores given by an arbitrary linear scoring function can thus be materialized.

Before materialization, tuples that are dominated by more than  $k$  tuples are discarded because they do not belong to the top-k query answer of any linear scoring function. The remaining tuples, denoted as the *dominating set*  $D_k$ , include all possible top-k answers for any possible linear scoring function.

Materialized views have been studied in the context of top-k processing as a means to provide efficient access to scoring and ordering information that is expensive to

gather during query execution. Using materialized views for top-k processing has been studied in the PREFER system [4, 5]. The score of a certain tuple is captured by an arbitrary weighted summation of the scoring predicates. The proposed method keeps a number of materialized views based on different weight assignments of the scoring predicates. Specifically, each view  $v$  ranks the entire set of underlying tuples based on a scoring function  $F_v$  defined as a weighted summation of the scoring predicates using some weight vector  $\mathbf{v}$ . For a top-k query with an arbitrary weight vector  $\mathbf{q}$ , the materialized view that best matches  $\mathbf{q}$  is selected to find query answer. Such view is found by computing a position marker for each view to determine the number of tuples that need to be fetched from that view to find query answer. The best view is the one with the least number of tuples to be fetched.

### 3.2.2 Engine Level

The main theme of the techniques discussed in this section is their tight coupling with the query engine. This tight coupling has been realized through multiple approaches. Some approaches focus on the design of efficient specialized rank-aware query operators. Other approaches introduce an algebra to formalize the interaction between ranking and other relational operations (e.g., joins and selections). A third category addresses modifying query optimizers, e.g., changing optimizers' plan enumeration and cost estimation procedures, to recognize the ranking requirements of top-k queries.

Query optimizers need to be able to enumerate and cost plans with rank-aware operators as well as conventional query operators. Costing a rank-aware operator is quite different from costing other traditional query operators because a rank-aware operator is expected to consume only part of its input. Furthermore, the size of the consumed input depends on the operator implementation rather than the input itself. A probabilistic model has been proposed in [8] to estimate the Rank-Join inputs' depths, i.e., how many tuples are consumed from each input to produce the top-k join results. A related approach has been further pursued in [14].

## 4 Optimization of Rank-Join Algorithms

In this section we discuss several optimization requirements and opportunities in the context of rank-join algorithms for search computing. First, in Section 4.1, we reconsider some of the challenges described in Section 2 and propose corresponding approaches. Then, in Section 4.2, we revisit cost models used for optimization and, finally, in Section 4.3, we discuss possible adjustments of strategies when the actual statistics extracted during the execution of rank join differ from those estimated beforehand.

### 4.1 The Need for Query Optimization in Search Computing

Among the main challenges for rank-join in Search Computing, we mentioned the need to obtain query answers quickly by limiting the number of requests sent to services to retrieve data and compose combinations, yet guaranteeing that the top k combinations are formed. We can characterize such a successful execution of a

rank-join between two services, say  $s_1$  and  $s_2$ , with a “descent” retrieving  $n_1$  and, respectively,  $n_2$  tuples. In this respect, we can express the expected number of formed combinations  $K(n_1, n_2)$  and the expected cost  $C(n_1, n_2)$  incurred by such a descent as functions over  $n_1$  and  $n_2$ . In particular,  $K(n_1, n_2)$  will relate the numbers of tuples retrieved with available information about the join selectivity, the distribution of values in the data returned by  $s_1$  and  $s_2$ , and, possibly, the score distributions in  $s_1$  and  $s_2$  as well as the aggregation function used to compose the individual scores; similarly,  $C(n_1, n_2)$  will take into account the costs of accessing  $s_1$  and  $s_2$ . Consequently, the optimization problem at hand may be formulated as follows:

$$\begin{aligned} &\text{minimize} && C(n_1, n_2) \\ &\text{subject to} && K(n_1, n_2) \geq k \\ &&& n_1 \in \mathbf{N} \cap [0, N_1], n_2 \in \mathbf{N} \cap [0, N_2] \end{aligned} \quad (1)$$

where  $\mathbf{N}$  is the set of natural numbers,  $k$  is the desired number of top combinations, and  $N_i$  is the maximum number of tuples that can be output by  $s_i$  for  $i=1,2$ . A solution to the above optimization problem will provide estimates for  $n_1$  and  $n_2$  and, as a result, an estimate of the cost incurred by the rank-join operation.

Another challenge posed in Section 2 regarded the ability to quickly respond, by updating the results, to changes in user’s requirements and needs, as will be further discussed in Chapter 13 about “liquid queries”. The notion of liquid queries includes user’s feedback and the ability to adjust several parameters concerning the query. Among the available operations, the user can request a “query re-ranking”, i.e., an update of the results of the query by modifying the aggregation function, e.g., by changing the weights of the individual scores when the aggregated score is expressed as a weighted sum thereof. Such an operation is not equivalent to a simple rearrangement of the combinations obtained with the previous aggregation function, but generally requires computing a new and different set of combinations, possibly by deepening the previous descent to the services. A possible approach to query re-ranking consists then in computing, from the beginning, a set of combinations that is comprehensive enough to be “robust” with respect to changes in the aggregation function. In other words, instead of limiting ourselves to computing a set of combinations that includes the top  $k$  ones with respect to a given aggregation function, it might be preferable to compute a larger set that includes the top  $k$  combinations no matter what the aggregation function is. This allows one to simply sort the combinations within the set according to the new aggregation function, without the need to access the services again after a change in the aggregation function. When both sorted and random accesses are available, this behavior may be obtained, e.g., by adapting to the rank-join case the original rank aggregation algorithm by Fagin (called  $A_0$  and better known as Fagin’s algorithm [16]).

## 4.2 Cost Models for Query Optimization

In order to solve the query optimization problem formulated above, which refers to a single rank-join operator on two services, we need to define a cost model that characterizes the cost function  $C(n_1, n_2)$ . Most of the previous literature on rank-join

adopts a simple additive model, whereby the cost is defined as the sum of the costs of all I/O operations. Both sorted and random access (whenever available) costs need to be taken into account, since they are possibly characterized by heterogeneous costs, due to the fact that random accesses might potentially refer to data that is stored in other external data sources. While this approach is still applicable in the context of search computing, we want to take advantage of the fact that services are typically available at remote servers. Therefore, more flexibility is given in the way services can be invoked, i.e. by exploiting parallel invocation. Nevertheless, parallelism affects both the actual execution strategy and the cost model that drives the query optimization, and needs to be carefully addressed.

So far, we have discussed rank-join operations involving two services. Answering complex queries might require combining multiple operations (both rank-join and other operations) into a query plan. Thus, query plan optimization entails determining the sequence of execution among the operations, as well as, in the case of rank-join, the number of tuples that presumably have to be fetched in order to match a target number of combinations. Global query optimization in the context of search computing is addressed in Chapter 10.

### 4.3 Need for Adaptive Algorithms

Before query execution starts, the optimizer uses the available information collected about the search services to be joined, in order to devise: i) a cost model, as explained above, which determines the number of tuples to be fetched; ii) the rank-join execution strategy, also called pulling strategy, which determines the optimal order of service invocations during the “descent” in the two services. It might be the case that statistics collected beforehand, e.g. join selectivity, score distributions, etc., do not match the actual fetched data. This is particularly relevant in the context of search computing, where the actual statistics depends not only on the query prototype, but also on the specific user provided keywords. Consider, for example, the different score distribution (e.g. when scores are indicated by prices) when querying for movie theatres in a large city as opposed to a small town. In this case, the query execution might be adaptively adjusted in such a way to take this into account. In concrete terms, this entails adapting the pulling strategy at execution time. We refer to this feature as intra-operator adaptive execution, as it involves only the internal machinery of an individual rank-join operator.

A different kind of adaptive execution might arise when considering complex queries in the large, i.e. in the context of a global query optimization framework. We refer to this feature as inter-operator adaptive execution. In this scenario, adaptive execution might be necessary in order to cope with time-varying availability of search services. During query execution, search services might stop responding to invocations and alternative execution strategies might be devised on-the-fly, possibly preserving the data already fetched up to that point in time. Similar issues arise when the response time, e.g. the cost of accessing a service, is highly non-stationary. Again, the query execution should be modified accordingly.

## 5 Conclusion

In this chapter we have introduced the rank-join problem and highlighted its role in search computing. We have described new challenges for rank-join regarding adaptivity, use of feedback, and new cost models. We have also given hints at the integration of rank-join in a global query optimization framework, as the one that was described in the previous chapter.

## References

- [1] Bruno, N., Chaudhuri, S., Gravano, L.: Top-k selection queries over relational databases: Mapping strategies and performance evaluation. *ACM Transactions on Database Systems* 27(2), 153–187 (2002)
- [2] Bruno, N., Gravano, L., Marian, A.: Evaluating Top-k Queries over Web-Accessible Databases. In: *Proceedings of ICDE 2002*, pp. 369–378 (2002)
- [3] Fagin, R., Lotem, A., Naor, M.: Optimal aggregation algorithms for middleware. *Journal of Computer and System Science* 1(1), 614–656 (2001)
- [4] Hristidis, V., Koudas, N., Papakonstantinou, Y.: PREFER: A System for the Efficient Execution of Multi-parametric Ranked Queries. In: *Proceedings of ACM SIGMOD 2001*, pp. 259–270 (2001)
- [5] Hristidis, V., Papakonstantinou, Y.: Algorithms and applications for answering ranked queries using ranked views. *VLDB Journal* 13(1), 49–70 (2004)
- [6] Ilyas, I.F., Aref, W.G., Elmagarmid, A.K.: Supporting top-k join queries in relational databases. *VLDB Journal* 13(3), 207–221 (2004)
- [7] Ilyas, I.F., Aref, W.G., Elmagarmid, A.K., Elmongui, H.G., Shah, R., Vitter, J.S.: Adaptive rank-aware query optimization in relational databases. *ACM Transactions on Database Systems* 31(4), 1257–1304 (2006)
- [8] Ilyas, I.F., Shah, R., Aref, W.G., Vitter, J.S., Elmagarmid, A.K.: Rank-aware query optimization. In: *Proceedings of ACM SIGMOD 2004*, pp. 203–214 (2004)
- [9] Li, C., Chang, K.C.-C., Ilyas, I.F., Song, S.: RankSQL: query algebra and optimization for relational top-k queries. In: *Proceedings of ACM SIGMOD 2005*, pp. 131–142 (2005)
- [10] Marian, A., Bruno, N., Gravano, L.: Evaluating top-k queries over web-accessible databases. *ACM Transactions on Database Systems* 29(2), 319–362 (2004)
- [11] Natsev, A., Chang, Y.-C., Smith, J.R., Li, C.-S., Vitter, J.S.: Supporting Incremental Join Queries on Ranked Inputs. In: *Proceedings of VLDB 2001*, pp. 281–290 (2001)
- [12] Schnaitter, K., Polyzotis, N.: Evaluating rank joins with optimal cost. In: *Proceedings of PODS 2008*, pp. 43–52 (2008)
- [13] Tsaparas, P., Palpanas, T., Kotidis, Y., Koudas, N., Srivastava, D.: Ranked Join Indices. In: *Proceedings of ICDE 2003*, pp. 277–286 (2003)
- [14] Schnaitter, K., Spiegel, J., Polyzotis, N.: Depth estimation for ranking query optimization. In: *Proceedings of VLDB 2007*, pp. 902–913 (2007)
- [15] Ilyas, I.F., Beskales, G., Soliman, M.A.: A survey of top-query processing techniques in relational database systems. *ACM Comput. Surv.* 40(4) (2008)
- [16] Fagin, R.: Combining Fuzzy Information from Multiple Systems. *J. Comput. Syst. Sci.* 58(1), 83–99 (1999)

- [17] Clough, P., Sanderson, M., Abouammoh, M., Navarro, S., Lestari Paramita, M.: Multiple approaches to analysing query diversity. In: Proceedings of ACM SIGIR 2009, pp. 734–735 (2009)
- [18] Gollapudi, S., Sharma, A.: An axiomatic approach for result diversification. In: Proceedings of WWW 2009, pp. 381–390 (2009)
- [19] Agrawal, R., Gollapudi, S., Halverson, A., Ieong, S.: Diversifying search results. In: Proceedings of WSDM 2009, pp. 5–14 (2009)

# Chapter 12:

## Panta Rhei: Flexible Execution Engine for Search Computing Queries

Daniele Braga, Stefano Ceri, Francesco Corcoglioniti, and Michael Grossniklaus

Dipartimento di Elettronica e Informazione, Politecnico di Milano,  
Piazza Leonardo da Vinci 32, 20133 Milano, Italy  
{braga, ceri, corcoglioniti, grossniklaus}@polimi.it

**Abstract.** The efficient execution of data-intensive computations over services is a challenging task: data are retrieved from remote sources and therefore are not available in the query engine until after the execution of these calls, but the system must be inherently efficient thereafter, by guaranteeing that data is immediately cached and processed efficiently, according to the best query plan. In this chapter, we present a flexible execution model for search computing queries, named Panta Rhei. The proposed execution engine paradigm adopts the producer/consumer model and supports both data-driven and event-driven synchronization, and their interplay. Query plans are modeled as directed graphs, whose nodes are processing units and whose edges are either control or data flows. While control flows synchronize service calls and unit execution, data flows transfer data between units that process data flows to produce query results. We present the specification of Panta Rhei by formally defining the units for data production, consumption, manipulation, and caching, as well as the control and data flows. Finally, we discuss how a query plan is expressed in terms of a query execution plan.

### 1 Introduction

Query execution in Search Computing is a data-intensive process. The computations required for answering a query, although performed upon the data resulting from service calls, are very similar to those performed by database management systems working on physically optimized tables. Therefore, a query execution engine supporting Search Computing must be able to efficiently support dynamic data extraction, storage and caching, as well as efficiently route data flows between special-purpose computational units, whose design has been optimized so as to guarantee the fast production of query results.

Due to the very nature of many of these tasks and their embedding within Web-based contexts, which are subject to continuous change, performances of data-intensive service interactions are very hard to predict. Moreover, the execution engine must be strongly connected to the query user interface, so as to adapt to user requests that dynamically alter the query requirements, either by specializing current requests or by adding new requirements. For these reasons, the design of the query execution engine for Search Computing has required several architectural solutions for supporting dynamic adaptation which are quite original, especially for what concerns the synchronization aspects.

The main operation in Search Computing is the join of search services and, therefore, the execution engine is optimized to support joins, under the constraint that join data operands are not immediately available to the execution engine, but are produced by interacting with services, ranked and separated in chunks. Join processing, as explained in the previous chapter, aims at exploring given compositions of chunks returned by the services. In this setting, optimization consists in minimizing the number of service calls and, at the same time, in efficiently exploring the search space so as to rapidly produce results.

Supporting join executions requires synchronizing pairs of services. To effect this synchronization, we introduce particular units, called clocks, whose effect is to give pulses to services so as to synchronize them according to certain mutual relationships that can be dynamically adapted. In order to respond to variability, synchronization is subject to feedbacks which are generated within the execution environment. The explicit (and user controllable) synchronization and adaptation of join computations through clock units is the most significant (and original) aspect of the execution engine, being used both for pipelined and parallel execution with a uniform style.

Original aspects of the execution engine concerns the explicit management of chunks within the data flow, which is at the basis of the design of both the chunker units (capable of changing the size of chunks along the data flow) and the cache units (which store the results of service calls by chunks). In SeCo joins, a given chunk of a service's results can be involved in many chunk combinations, performed after its initial loading, and cannot be discarded until query processing is completed. Chunk support allows for an intermediary granularity level, which is a good compromise between tuple-level (each tuple flows individually) and table-level (each data collection or table flows as a unit) granularity. We believe that this solution yields to a good trade-off between flexibility, adaptability, and performance.

While clocks and chunks are, therefore, the main ingredients of the flexible execution engine, many other features characterize its design. The system must, of course, support sorting (i.e. ranking of results) which is a critical operation, because it is "blocking" (in order for the sort to be applicable to a given collection, all the items of the collection must be available) and data flow machines must try to minimize blocking operations. In addition, the system should support the early evaluation of selection predicates in order to reduce the size of data flows.

The organization of this chapter is as follows. In Section 2 we present the state-of-art of data-driven execution engines, first by highlighting the issues which arise in interpreted environments (such as ours) and then by focusing on adaptability of computations, the main quality offered by *Panta Rhei*. Section 3 presents the model, with its nodes representing units and edges representing data and control flows. Then, Section 4 sketches the translation of query plans into query engine execution plans, and Section 5 shows the typical translations of parallel, pipe, and top-k joins into schedules.

## 2 State of the Art

This section gives an overview of the state of the art of query execution with a focus towards the domain of Search Computing. First, we discuss different query processing



paradigms that serve to position the proposed query execution environment for Search Computing. A distinguishing feature of a query evaluation paradigm is the degree of query plan adaptation that is supported by an execution environment. In the second part of this section, we motivate the need for query plan adaptation at run-time and give an overview of related work on adaptation in other application domains.

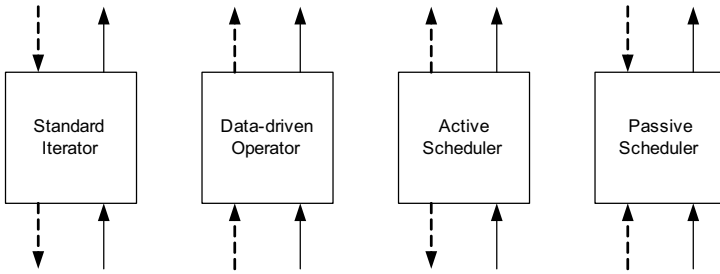
## 2.1 Query Processing Paradigms

An important criterion in the design of a query execution engine for Search Computing is its query processing paradigm. In the past, several types of query execution engines have been proposed in the scope of traditional DBMS, such as interpreted or compiled [19] execution engines. On the one hand, interpreted approaches translate queries into query plans that are optimized and evaluated leveraging a general-purpose set of operators provided by a virtual machine such as the query evaluator of a database management system. Compiled approaches, on the other hand, use code generation to translate each query into a static program that is compiled and executed natively, i.e. directly on the operating system. The main strength of compiled engines is their performance as all meta-information required for evaluating a query is directly hardwired into the program code. The gain in performance comes, however, at the price of flexibility. While compiled engines are fast, it is more difficult to cater for run-time adaptation of query plans as this would require a recompilation of the program while it is executing. Due to the requirements of Search Computing, we have, therefore, chosen to build an execution environment that follows the approach of an interpreted engine, and therefore we focus this state-of-the-art on interpreted query engines.

Interpreted engines can be further classified according to the query evaluation model that they use. Within interpreted engines, query execution plans require both control flow, which dynamically defines how engine modules are synchronized, and the data flows, which dynamically define data exchanges. From the viewpoint of data flows, components are characterized as producers and consumers and a query computation may involve several modules. At its beginning, a query plan involves producer modules, later intermediate components play both roles, and eventually query interfaces present their results to the user who is the “final consumer” of the system. Execution plan components, or “nodes”, have four possible behaviors relative to control and data flows, presented by Graefe (see [12], p.149ff) and shown in Fig. 1.

- *Standard iterators*. In most query processing systems, the data flow is demand-driven and controlled by the consumer. In this case, control and data flow point into opposite directions. According to [17], most state of the art approaches for *distributed query processing* use the iterator model [13] in which all operators exhibit an *open()-next()-close()* interface.
- *Data-driven operators*. There are however systems such as real-time or data stream systems where the data flow is paced by the producer as it needs to unload the data as it arrives, e.g. sensor data. In a data-driven operator, control and data flow point into the same direction.

To combine demand-driven and data-driven operators, it is necessary to introduce *flow translation nodes* [12] that mediate between the two types of operators.



**Fig. 1.** Nodes of an execution plan with control and data flow [12]; control flows are dashed arrows and data flows are solid arrows

- *Active scheduler.* A data flow translation node that can be used to schedule a demand-driven operator (iterator) as producer and a data-driven operator as consumer. This node actively requests data from a demand-driven producer and passes it on to the data-driven consumer.
- *Passive scheduler.* A data flow translation node that can be used to schedule a data-driven operator as producer and a demand-driven operator as consumer. As soon as the data-driven producer delivers data, this node accepts and buffers it until the data is requested from the eventually resumed demand-driven consumer.

Another important characteristic of the query processing paradigm is whether the execution is governed *centrally* by a global scheduler that has complete knowledge or in a *distributed* setup where the nodes of the execution engine make local scheduling decisions based on incomplete knowledge. For example, data flow systems [25] and stream processing systems, e.g. Aurora/Borealis [1], have addressed the problem of scheduling data-intensive computations. More recently, scheduling algorithms have been proposed that control the execution of a computation in peer-to-peer networks, such as the economic model [2] or approaches based on reinforced learning [20]. While data flow systems are designed for the execution of fine-grained computations, workflow systems [27] address coarser-grained processes executed over Web services. While these two families of systems are similar in terms of goals, the latter tends to use central scheduling that operates on global information.

## 2.2 Adaptation

The capability of adapting software systems to internal or external requirements is often referred to as *adaptation*. Adaptation can be effected at design or compile-time of a system as well as at run-time. The need for adaptation is present in many application domains and adaptation can be supported at very different levels of granularity. Therefore, the entire body of research on adaptation is very vast and its complete review out of the scope of this chapter. Instead we will limit the discussion to work that is relevant in the context of query execution in Search Computing and structure them according to the scope of their application, from coarse-grained to fine-grained. We will start with adaptation at the level of the architecture, then discuss the adaptation of applications and conclude by presenting solutions to adapt processes in particular data-driven computations.

On an architectural level, the focus on adapting processes lies on leveraging the resources that are at disposition best. To that end, load-balancing schemes or the dynamic reassignment of resources to computation nodes are techniques that are often employed in the area of distributed computing, such as grid or cloud computing. On a level of finer granularity, we note that the adaptation of applications is a requirement that frequently arises in mobile and ubiquitous computing as well as in Web engineering. Context-awareness is a solution often proposed to adapt applications to limited device resources, environmental factors or multiple output channels. In the area of context-awareness, work has also been done on context-aware data management and querying [14]. Finally, it is also possible to perform adaptation on the level of individual computations that can be both process-driven and data-driven. In the following, we will focus entirely on adaptation of data-driven computations such as query plan adaptation since this is most closely related to the query execution engine presented in this chapter.

Generally, query plan adaptation can be classified according to when it is taking place into *compile-time* and *run-time* adaptation. On a finer level, query plan adaptation can be refined further according to the information that is used as input to effect the optimization. We distinguish the types of input information given below and, in the following, discuss how each one of them can be used for adaptation.

- *Data statistics* such as the cardinality of tables and the selectivity of predicates.
- *Usage statistics* obtained through profiling of query execution or mining of query execution history to get dynamic statistics (self-tuning databases).
- *User control* that determines the adaptation of the query plan.

Clearly, some of these types of information are mostly used for compile-time adaptation, while others only make sense for run-time optimization. For example, data statistics are usually leveraged at compile-time by the optimizer to plan the execution of the query in the best possible way. While usage statistics are typically gathered at run-time, either using “pay-as-you-go” frameworks [5] or in separate mining processes, this information is also applied to the adaptation of the query plan at compile-time. As a consequence, the queries that are profiled or mined do themselves not profit from this information as only later executions of the same or similar queries are adapted accordingly.

Nevertheless, approaches that use data and usage statistics for supporting the dynamic reoptimization [16] and adaptation of query plans exist. Among those approaches are adaptive operators, query scrambling, the interleaving of query planning and execution, and opening up the query optimizer to application input. Adaptive operators, such as e.g. *choose nodes* [6], *XJoin* [23], or *BindJoin* [18], are query plan nodes that defer certain decisions until execution. In the former case, choose nodes select at run-time from a set of query sub-plans that was defined at compile-time. In the latter cases, the join implementations themselves are capable of adapting to delays at run-time. Another more dynamic approach for dealing with unexpected delays is *query scrambling* [24] that modifies the query execution plan on the fly based on heuristics. In approaches that use *interleaving*, e.g. [26] or [8], the optimizer only produces a partial plan for the execution engine and decides how to proceed once that partial plan has been evaluated. Finally, the author of [4] argues that future query optimizers should also benefit from rich usage data and application

input. Adaptive query execution systems for *data integration over the Web* address the problems of absence of statistics and unpredictable data arrival characteristics. Most of these systems combine novel approaches, e.g. incomplete query plans that are completed and (re)optimized incrementally [15] with existing concepts such as the previously discussed interleaving of query planning and execution as well as adaptive operators. Adaptive query processing approaches that leverage information captured through self-monitoring of the query execution have also been proposed for Grid computing [11].

Finally, *user control* as an input for process adaptation has been addressed in systems that allow performance and query execution to be expressed through interactive dashboards [7]. As most work on dashboards has been done by the HCI community, it largely addressed the interface level in terms of visualizing complex and large sets of information in a comprehensive and graspable way. Nevertheless, there are also approaches that focus on the evaluation of queries in the presence of a visual and interactive interface. For example, [22] shows how dynamic query interfaces can be supported in large databases through the use of incremental data structures and algorithms. The approach introduces the notion of an active subset of the database that is enhanced with auxiliary data structures designed to support continuous querying. These auxiliary data structures are directly coupled to the interface and are only reprocessed in the event of user interaction. Results are visualized incrementally by computing and displaying the delta resulting from the user input. In [3], a classification and survey of visual query systems for databases is presented.

### 3 Panta Rhei Specifications

While classic execution engines operate upon databases which are initially stored within the memory (possibly distributed and replicated), query execution in Search Computing requires the efficient execution of joins between results of service invocations and, hence, the main flows of data production fall outside of the engine's control. The need of combining service invocations with data-intensive operations is the main architectural challenge, approached by a modular decomposition of the process into processing units and by an explicit description not only of the data flow, as it is typical of many run-time architectures, but also of the control flow, through dedicated units and signals. Control flow modeling enables to explicitly tune execution, adapting it to unexpected behaviors of the components.

This concept is illustrated in Fig. 2. In the plan, the input unit, after its activation at query start, sends a control pulse to a search service unit, which executes a call. The call's result is a data flow which is sent to the output unit and, hence, returned to the query interface.

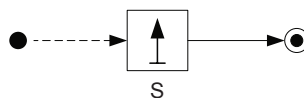


Fig. 2. Simple execution plan

In the following, control flows, data flows, and processing units are described in detail.

### 3.1 Structure of Execution Plans

Panta Rhei is a dedicated environment for the processing of execution plans. Every execution plan represents the physical evaluation of a query plan, and consists of a directed graph of nodes (units) and edges (data and control flows), where

- The *data flow* is a directed acyclic graph connecting processing units and whose closure defines a precedence relationship between units (the “flow of execution”). The data flow itself consist of chunks of combinations (tuples) which are progressively created by joining pairs of services, therefore in the end the flow of executions produces the result tuples. Search computing results are duplicate-free, and therefore once a tuple is formed along the dataflow, another identical tuple can be removed from the computation.
- The *control flow* includes pulse signals which are propagated “forward” (i.e. along the flow of execution) in order to time and synchronize service calls, and suspend/resume signals which are propagated “backward” in order to re-synchronize execution when anomalies are detected. Therefore, the forward controls determine producer-consumer relationships according to the query plan, and the backward controls optionally conditions those producer-consumer relationships that deviate too much from the optimal plan determined at query optimization time. A control edge may start from a data producer and, in this case, every new chunk of data produced by the unit also produces a new pulse signal.
- The behavior of each node is completely determined by its input and state. Some units accept at most one input pulse, if the pulse is omitted then the unit responds just to data flows. All nodes receive their data input from one predecessor, with the exception of parallel joins and cache units, that can have more than one data flow edges as input, as they implement binary operations (join and union).

Query plans include parallel and/or pipe joins (as presented in Chapter 10) which are translated into nodes of the execution plan. While a pipe join is represented as a sequence of service calls in which the second call implements the join, a parallel join requires an explicit join unit which has two service units as predecessors. The parameter setting of nodes involved in join computation is optimized according to the service interface specifications (particularly, their chunk sizes and service costs). The translation of an optimal query plan into its execution plan is rather straightforward, as the topology of the execution plan can be immediately drawn from the query plan. Instead, the initialization of node parameters dictating the specification of the operations implemented by them is not covered in the book. At the moment, we use simple heuristics to initialize the parameters, but we expect to fine-tune the heuristics after experimentation.

Conceptually (the implementation may be different), each node is mapped to a thread which is activated at query start, waits for input, and produces output. Queries can be suspended and resumed by users according to the liquid query interface controls, described in the next chapter. At query start (or resume), some user-controlled

parameters may be fetched into appropriate “slots” of units to fully specify their behavior. Most of these parameters are defined by the query optimization process. Then, the start node of the execution plan is activated, which triggers the start of its successor nodes. Nodes either act as data producers or consumers, or play both roles. During the execution, data producers can send “EOF” data along one data flow link, with the semantics that there will be no more data along the link. The “EOF” data is propagated by consumers until it reaches the output node, causing query termination and then the output to be produced.

The liquid query interface communicates with the execution engine by various controls, and the effect of controls may suspend or terminate a query execution. Users may also change the content of some of the query “slots” which are exposed to the user interface (through user-friendly formats). Threads are eliminated only when the user “changes” the query. Memory caches, however, might be emptied if the user “repeats” a query with a different input.

### 3.2 Scheduling Units in *Panta Rhei*

The semantics of execution plans is rather complex, as it requires introducing a number of ingredients (concerning units and their control) which interact with each other. We have decided to first present all ingredients and then to show their interplay through examples. The types of flows offered by the execution engine have already been introduced above and we recall that control flows are signals carrying no information other than their intrinsic nature (pulse, suspend, resume), while data flows carry chunks of tuple combinations, made up by matching results of the previous service calls in the flows, and emitted by units in chunks, according to the unit’s execution semantics. In the following, we therefore focus on an in-depth presentation of the various kinds of units, which are shown in Fig. 3.

**Input and Output Units.** The *input unit* injects user-provided input into suitable slots of given units, and then starts the execution. Each execution plan must have exactly one input node, which has one or more successor nodes.

The *output unit* is a consumer node collecting query results. Each plan must have exactly one output node. Its execution activates the *liquid* interface showing the query results.

**Clock Unit.** A *clock unit* plays the role of coordinating service calls to perform pipe and parallel joins – thus, it is neither a producer nor a consumer. Topologically, every clock unit has in its children at least two service calls. Every clock in a plan controls a sequence of joins, where each join in the sequence is either a pipe or a parallel join, and the topology of the execution plan indicates the operands of each join<sup>1</sup>.

Every clock is activated by a start pulse signal (a control edge connects the input node to the clock) or by a data-producer unit which produces its first data (in this case, a control edge connects the data producer unit to the clock). Clocks emit *pulse*

---

<sup>1</sup> Currently, we associate every query with exactly one clock unit controlling all of its joins, but we plan to experiment with more general settings. As clocks can be activated during the execution flow, the semantics of clocks, service, and join units in the context of scheduling plans does not force plans to have a single clock.

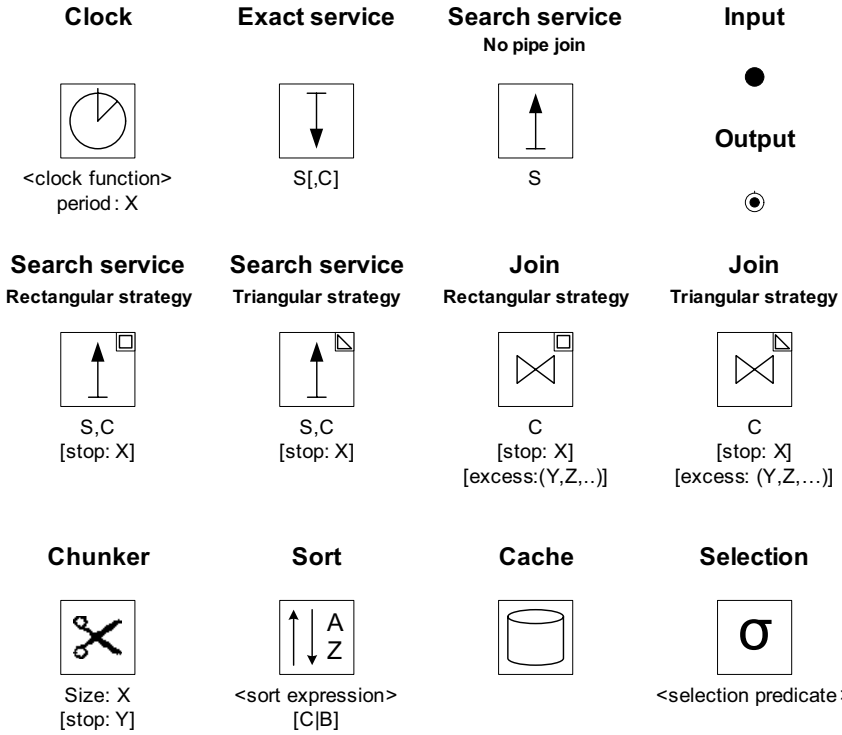


Fig. 3. Nodes of execution plans

signals to two or more service invocation units, ordered from 1 to  $n$  (the order of controlled units is given by the numbers labeling the pulse edges).

Every clock has a parameter, called *clock function*, defined by a regular expression that describes the maximum number of calls that the service unit can perform during the clock cycle, for each service unit controlled by the clock and for each clock cycle. To do so, the regular expression defines a sequence of clock values which each correspond to one clock cycle. Clock values are denoted as  $n$ -tuples of integer numbers, where  $n$  is the number of service invocation units controlled by the clock unit. Enumerable repetitions of sequences are indicated by a superscript, while infinite repetitions of the last parenthesis are denoted by an “ $n$ ” superscript. As an example,  $(1,1)(2,2)^n$  represents a sequence in which two services are invoked once in the first clock cycle, and then can be invoked at most twice in any subsequent clock cycle. An example of clock function for controlling three service units is:  $(3,1,2)(4,0,3)^2(5,1,4)^n$ . The clock function can be replaced at runtime, e.g. based on user input.

Every clock has a given *clock frequency* (cycles per second), which determines the time interval between two consecutive pulses to the descendent units. The clock frequency should be related to the average response time of the search services controlled by the clock. A reasonable recommendation is to set the frequency to cater for the execution of the largest of  $N_{ij} \times ART_i$ , where  $N_{ij}$  is the number of calls that unit  $i$  is enabled to perform during cycle  $j$  and  $ART_i$  is the average execution time of

service  $S_i$ . This number represents an estimate of the execution time due to the slower service which is controlled by the clock. Otherwise the clock would “enforce” a speed greater than the speed of one controlled service (and as such it can hardly impose any synchronization)<sup>2</sup>.

A clock can be *suspended* by any service that it schedules. Services may be slower or faster in producing tuples relative to the plan, and thus the joiner could deviate from the configured ratio. The rationale of suspend/resume is that triangular or rectangular strategies of joins should be faithfully implemented, as they were decided by an optimizer at compilation time by taking into account the features of the services (and attempting a minimization of their access costs), and thus deviations from plans occurring at run-time should be limited. A given amount of permitted deviation (ranging between zero and infinite deviation) is defined as a join parameter. If the allowed deviation is overcome, then the clock is suspended. As a consequence, the clock will not issue any more pulses until it will receive a resume signal, which in turn is sent by the same join unit when the deviation is reduced to an acceptable amount.

**Exact and Search Service Units.** *Exact service* units produce a finite set of tuples that represent the exact (and thus complete) response to the service call query given the input parameters. The output tuples are neither ranked nor chunked. Nevertheless, exact services produce sequences of chunks, where each chunk corresponds to one service invocation. In the context of pipe joins, these sequences may be ordered in the data flow due to their composition with previous calls to search services. Exact services are triggered by a single input, either a pulse or a data chunk.

- In the first case, denoted as *pulse input*, the pulse produces a single exact service call, performed as soon as the pulse is received. Therefore, a well formed graph should only allow pulse signals to an exact service with the “number of invocations” parameter set to one, which is assumed as default. If an exact service is called only once and independently of data flows, normally its input is filled by “slots” extracted from the query. This situation occurs when the exact service call does not depend on other services. Note that further pulse signals, in this case, should not be allowed by a correct graph, and anyway will have no effect on the unit (i.e. the service call will not be repeated). An EOF marker indicates the end of tuples in the result.
- In the second case, denoted as *chunk input*, the service triggering produces as many calls as there are tuples in the data chunk, performed as soon as the chunk tuples become available and continued until all tuples are consumed. In this situation, the exact service unit implements a pipe join, whose strategy is however rather simple, because it consists in a simple call iteration. The input parameters of each iteration are extracted from the input tuple, and the corresponding result tuples are combined (joined) with the input tuple, thereby producing an output chunk. If the input dataflow is ordered, then the chunks are produced by the service according to the input order.

---

<sup>2</sup> Setting the clocks’ frequency is a delicate service time vs. optimization time trade-off. Currently, we use as default solution setting the frequency exactly to the largest  $N_{ij} \times ART_i$  computed on all the clock’s edges.



*Search Services* exhibit a behavior similar to Web search engines: results are unbound, ranked and chunked, and normally there is no interest in obtaining a complete result, but only in obtaining the first chunks. They are triggered by either a pulse or a pair of data chunk and pulse.

- In the first case, denoted as *pulse input*, every pulse corresponds to a given number of service calls, progressively extracting new chunks at each call;  $N_{ij}$  denotes the number of allowed calls for service  $i$  in a clock cycle  $j$ . Normally, input fields for the call are filled by “slots” extracted from the query. This situation occurs when the search service call does not depend on other services.
- In the second case, denoted as *pulse and chunk input*, the same number  $N_{ij}$  of service calls is allowed as in the first case, but these calls can either use a new tuple from the input flow to match it with the “first” chunk of results for that tuple, or instead continue with another input tuple  $T_i$  that was already used in previous calls (thereby producing a given number  $C_i$  of chunks) and produce one or more subsequent chunks for that tuple (i.e. chunks starting with  $C_i+1$ ). This is the most complex case of pipe join strategy, which iterates over either new or already considered input tuples (which may be unordered or ordered by the first service call) and produces chunks (which, for each input tuple, are relatively ordered by the second service call). A pipe join strategy is used for choosing at each step, which follows either a rectangular or triangular strategy which will be discussed below. In any case, results are produced by chunks (whose size is given by the number of matching tuples produced at each call of the service) and the chunks are ordered.

Pipe joins occur when a dataflow input edge comes into a service call unit of arbitrary nature (either exact or search). A pipe join implements the join between services when the join attributes of the first service are *bound* and the join attributes of the second one are *free*. If the input data consists of the concatenation of  $N$  tuples, then the output data will consist of concatenation of  $N+1$  tuples, possibly represented through their keys. If either the input is ordered or the service being called is a search service, then the join output will be ordered.

When the second service being called is a search service, a *pipe join strategy* is needed to control the allocation of service calls to input tuples (as each input tuple is used to provide parameters for a service call, and the same tuple may induce several calls to the service, to find “better” combined results). The join strategy is imposed by performing a pipe join strategy on the downstream service unit, controlled by a clock, called the pipe join’s *clock controller*, whose clock function regulates the behavior of the two services. The input pulse parameters, sent by the clock controller to both services, indicate the number of calls allowed within a given clock cycle, and therefore also of chunks produced in output during a cycle. The pipe join strategy can be either rectangular or triangular, as informally represented in Fig. 4.

- In a *rectangular strategy*, the calls are performed considering every available input tuple in a round robin fashion: chunks are progressively extracted (the first chunk for tuples  $T_1, T_2, T_3\dots$  and then the second chunk for tuples  $T_1, T_2, T_3\dots$ ). This strategy is well suited when the first service is an exact service, producing unordered input. A rectangular strategy can be imposed by setting a parameter in

the second service (to  $R$ ) and timed by the join’s clock controller, by setting the clock’s function to  $(1,N)^M(0,N)^n$ , where  $N$  is the chunk size of the first service, and after the first  $M$  calls the first service produces an EOF. Then, calls have to be addressed just to the second service, producing the various layers of the rectangle, by iterating on the result tuples of the first service.

- In a *triangular strategy*, calls are performed in an alternate fashion: the first chunk is extracted for  $T_1$ , then the first chunk is extracted for  $T_2$  and the second chunk is extracted for  $T_1$ , and so on, as described in the right side of Fig. 4 (same as merge-scan parallel join). This strategy is well suited when the first service is a search service, producing ordered input. A triangular strategy can be imposed by setting a parameter in the second service (to  $T$ ) and timed by the join’s clock controller, by setting the clock’s function to the sequence  $(1,N)(1,2N)(1,3N)\dots$ , where  $N$  is the chunk size of the first service, thereby offering to the second service the option to get new chunks both for new tuples and for already available tuples of the first service.<sup>3</sup>

**Join Units.** Join units support the parallel join between two services, i.e. a join when neither of the join attributes is *bound*. A join unit joins the available information “by chunks”. Each chunk combination gives rise to a “tile” of results (i.e. tile  $(1,1)$ ,  $(1,2)$ ,  $(2,1)$ ,  $(2,2)\dots$ ), as discussed in the previous chapter. Therefore, it has as predecessor (at least) a pair of search services (producing chunked data). A *join strategy* specifies the order of exploration of tiles, with the aim to process tiles with higher rankings and more matches as fast as possible. A merge-scan strategy is obtained by setting the clock’s function to  $(N,M)^n$  where  $N/M$  is the optimal ratio between chunks of the two services. A nested-loop strategy is obtained by setting the clock’s function to  $(1,N)(1,0)^n$ , where  $N$  calls are required to exhaust the second service and then calls are

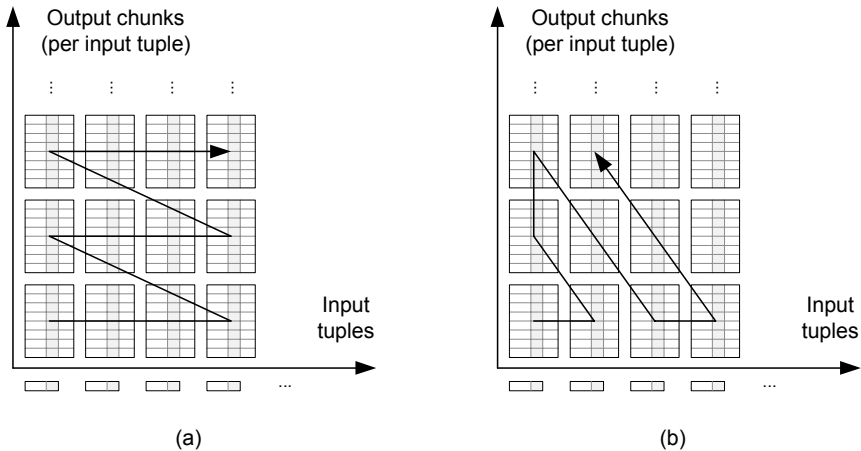


Fig. 4. Rectangular (a) vs. triangular (b) pipe join strategies

<sup>3</sup> A pure triangular strategy can be modified by defining “triangles” more properly, e.g. with an arbitrary proportional alternation. This extension is left for future work.

only addressed to the first service. In both cases, the choice of rectangular or triangular strategy is specified by setting a parameter in the join unit (to *R/T*). If either of the join input is not chunked, then the strategy is not needed and the parallel join degenerates to a unit which has a simple implementation, consisting in producing tiles in the only possible order (e.g. (1,1), (1,2), (1,3)...).

A join unit can have a stop parameter, which indicates the maximum number of tuples that should be produced by the join before suspending its execution (and producing an EOF marker)<sup>4</sup>.

In addition, a join unit has a pair of local parameters describing the amount of deviation allowed from the planned strategy. Deviation only occurs if the join greedily attempts to produce more chunks than the number allowed by available input and planned strategy. These numbers count how many additional input chunks can be joined from either services, ranging from (0,0) – no deviation for either services – to a given pair (2,3), to unspecified – no deviation control. When the maximum allowed deviation on one input (corresponding to a service running “too fast”) is overcome, the clock controlling the join unit is suspended. At that point, the service running “too slow” has some pulses available, and the join unit can concentrate upon the “tiles” which were left behind due to the slowest service, in order to bring the proportion of service calls back within the specified limits. Finally, at that point, the clock is resumed.

**Selection Unit.** A selection unit receives a (chunked) dataflow in input and produces a (chunked) data flow in output, consisting of all the tuples which satisfy a selection condition (an arbitrary Boolean expression of selection predicates). The selection unit does not re-chunk the output to a given chunk size and, thus, possibly changes the size of the chunks according to the selectivity of the predicate. Equality predicates matching input attributes to constants are used for building service calls, while a selection unit computes additional selections (e.g. comparison operations between attributes). Classical methods are used (by the query optimizer) in order to place the selection unit immediately after the join operation (either pipe or parallel join) which constructs the tuple with the attributes required for computing the predicate.

**Sort, Chunker and Cache Units.** A *sort unit* gets in input chunks of tuples and produces re-ranked result tuples in output, according to a sort expression. Sort units can be “continuous” (they sort the input chunks one by one, as they are available) or “blocking” (they wait for an EOF marker, and then process the whole input accumulated so far and emit the reordered tuples as a single output chunk). The sort function is a weighted sum of normalized expression (in the [0..1] range) over input tuple attributes, with a sort direction (either ascending or descending).

A *chunker unit* constructs new chunks from input tuples, by ignoring any already existing chunk structure thereupon. It is configured with a desired output chunk size. A chunker emits a new chunk as soon as there are exactly as many tuples as the chunk size. It has a “stop” parameter indicating the number of chunks it should produce before placing an EOF on its output dataflow, which can be interpreted by the output unit as the signal for producing output to the interface. A chunker is normally the last

---

<sup>4</sup> An EOF marker can be overridden by the user to resume the query plan and produce more results.

unit before the output unit and therefore the suspension stops the computations returning the control to the user, who in turn can resume computations and ask for more outputs.

A *cache unit* stores, within temporary memory, chunks of tuples, retrieved from services, or tuples of their keys forming combinations produced by joins. Conceptually, a cache unit is present after every service or join unit in order to store the service call or join results. However, in order not to overload the representation of an execution plan, we may omit cache units unless they have more than one incoming edge. In this case, all incoming tuples share the same schema and the cache implements the union of these tuples. The cache can also change the order of combinations when used as a union and, hence, its edges are labeled accordingly (e.g. S1/S2). The cache memory uses the normalized schema of the services in order to store service call results, and stores combinations as tuples of keys of the primary table of each service. The keys are system-generated and the tuples are indexed by chunk number and by key.

### 4 Examples

This section presents examples of execution plan models in increasing complexity. The purpose of the examples is to show, although on a limited sample, that execution plans can support various join strategies, including parallel join, pipe joins, and the Fagin join method which gives top-k guarantees.

We start with a parallel join of search services (Fig. 5), which is discussed at length in Chapter 10. The execution of a merge scan join between two services  $S_1$  and  $S_2$  using a connection pattern  $C_1$  requires a triangular join strategy. If the optimizer determines that the optimal ratio between calls to service  $S_1$  and  $S_2$  is  $1/2$ , it is sufficient to set the clock function to  $(1,2)^n$  which means that at each cycle  $S_1$  performs (at most) one call while  $S_2$  performs (at most) two calls. The clock frequency is set so that the slowest call sequence (e.g. the time required for completing either one call of  $S_1$  or two calls of  $S_2$ ) takes place within about one cycle. The join at each new iteration builds tiles in triangular fashion, e.g. first tiles (1,1), (1,2), then tiles (2,1), (2,2), (1,3), (1,4), and so on. The joiner is allowed to produce

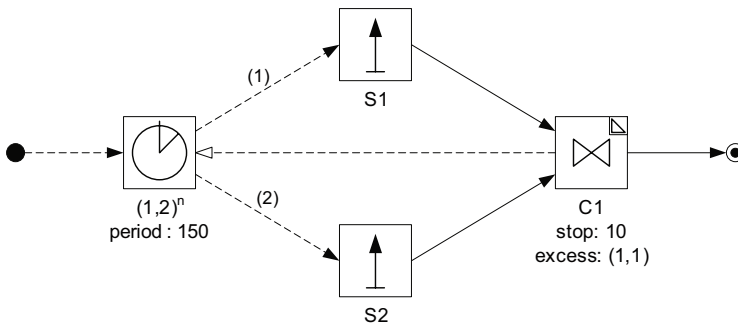
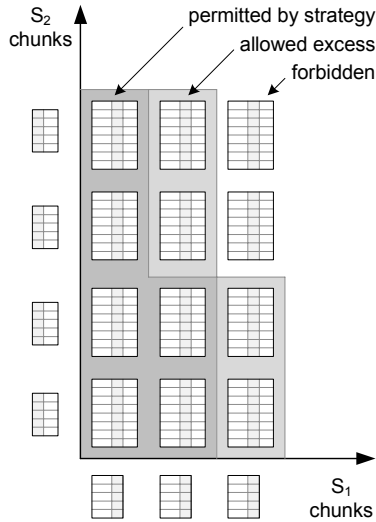


Fig. 5. Parallel join of two search services



**Fig. 6.** Join space permitted by the strategy and allowed excess for the parallel join of the example in Fig. 5, in case  $S_1$  returned 3 chunks and  $S_2$  returned 4 chunks

more tiles, e.g. if at a given point of time  $S_1$  has already produced 3 chunks and  $S_2$  has produced only 4 chunks, thus going beyond the  $1/2$  ratio, the joiner can proceed with the tile (3,1), (3,2) and then (2,3), (2,4), still keeping a triangular strategy. By doing so, it reaches its “allowed excess”, which is 1 extra-chunk (see Fig. 6).

In this case, if  $S_1$  produces one more chunk, the joiner signals the clock, and the clock in turn stops sending pulses until  $S_2$  produces 8 chunks, re-establishing the  $1/2$  ratio. Then, the joiner resumes the clock, and the execution of service calls and joins continues according to the joiner’s triangular strategy. The joiner is set to stop its execution, producing an EOF, when 10 result tuples are built. When the EOF is received by the output node, it presents 10 result tuples to the liquid query user interface.

We next illustrate a pipe join on the same services and connection pattern (Fig. 7). We implement a nested loop join, in which we assume that  $S_1$  produces chunks of size 10 and that after 5 calls it produces all relevant results. Then, the ratio between calls to  $S_1$  and  $S_2$  is  $1/10$  (every tuple of  $S_1$  is an input to  $S_2$ ) and the number of times this ratio must be iterated is 5. This enables building tiles (1,1)...(50,1), where each tile is obtained for a different tuple in input. At that point, no more calls to  $S_1$  are needed (all 50 tuples are cached) and therefore calls to  $S_2$  must be performed.  $S_2$  then performs the joins, thus producing the second, third, ..., and  $i$ -th chunk for the 50 cached tuples. The execution is terminated as soon as 20 result tuples are produced and an EOF is produced to transmit the result to the query interface through the output unit.

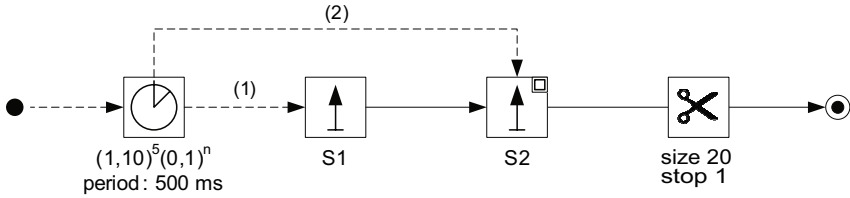


Fig. 7. Pipe Join of two search services

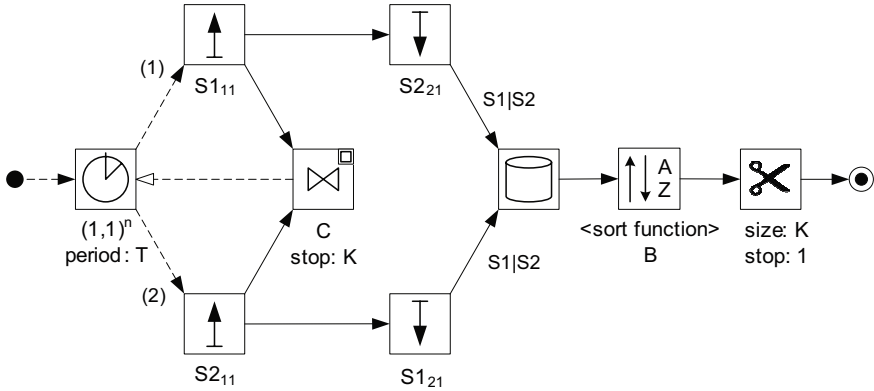


Fig. 8. Execution plan for a Fagin Join

The next example is the Fagin join [9] (Fig. 8). We recall that the method is applicable when both sequential (rank-based) and random (key-based) accesses are available for both of the services involved, and the method guarantees the extraction of *top-K* combination tuples, i.e. the tuples which are the best *K* according to any monotonic function of their relative rankings. We regard the Fagin method very suitable to Search Computing for this generality and for the method’s full definability at compile time, although it is suboptimal if compared with the threshold method, as discussed in Chapter 11.

Fig. 8 shows an execution plan for the parallel join of two search services  $S_1$  and  $S_2$  (supporting sequential access) followed by the pipe join of different service interfaces of services  $S_1$  and  $S_2$ , supporting direct access (e.g. access by an identifying property). A parallel join serves the purpose of halting the pulses to the search services as soon as *K* tuples are built. Then, by making a direct access for all join result values respectively on  $S_2$  – if the join value comes from  $S_1$  – and on  $S_1$  – if the join value comes from  $S_2$ . Results are then reordered and stored into a cache unit which performs their union. Eventually, results are sorted according to the sort function to obtain the single chunk of *K* resulting tuples, which are guaranteed to be *top-K*.

Finally, Fig. 9 shows an execution plan for the running example which queries for a good and recent adventure movie with screenings in a theatre not too far from the user’s home and good restaurants nearby. The clock controls in this case a parallel join which is followed by a pipe join. The parallel join combines *Movies* with

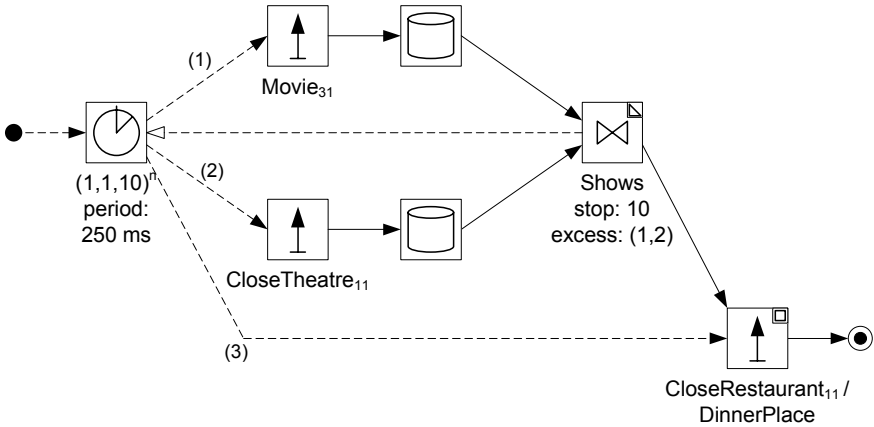


Fig. 9. Running Example

*CloseTheatres* according to the *Shows* combination pattern. The join combines one chunk of *Movie* with two chunks of *CloseTheatre*, using a triangular strategy, and with allowed excesses also set to (1,2). The join stops after producing 10 combinations of movies and theatres. Meanwhile, the data flow of the join results are sent to the *CloseRestaurant* service through the *DinnerPlace* connection pattern. For each pulse to *Movie*, 10 pulses are sent to *CloseRestaurant*, thereby enabling a tuple-based with 10 input tuples on from the first iteration, so that every matching movie-theatre pair is associated with close-by restaurant of the desired kind. Once 10 pairs are produced with a variable number of matching restaurants, execution is completed and results are transferred, through the output unit, to the user interface.

## 5 Conclusion

The execution engine described in this chapter supports operations such as service calls, join processing, caching, sorts, and chunking in order to support the efficient execution of the optimal plan selected by the optimizer. The execution engine prototype is a running platform which fosters the experimentation with new ideas and novel join strategies. Its extensible organization allows us to easily introduce new nodes or to change their parameters. The execution engine model is rather preliminary and will be improved while new releases of the environment will be deployed, yet the model resolves most of the technical challenges that are set by Search Computing queries.

## References

1. Abadi, D.J., Ahmad, Y., Balazinska, M., Cetintemel, U., Cherniack, M., Hwang, J.H., Lindner, W., Maskey, A.S., Rasin, A., Ryvkina, E., Tatbul, N., Xing, Y., Zdonik, S.: The Design of the Borealis Stream Processing Engine. In: Proceedings of Second Biennial Conference on Innovative Data Systems Research (CIDR 2005), Asilomar, CA, USA (January 2005)

2. Buyya, R., Abramson, D., Giddy, J., Stockinger, H.: Economic Models for Resource Management and Scheduling in Grid Computing. *Concurrency and Computation: Practice and Experience* 14(13-15), 1507–1542 (2002)
3. Catarci, T., Costabile, M.F., Levaldi, S., Batini, C.: Visual Query Systems for Databases: A survey. *Journal of Visual Languages and Computing* 8(2), 215–260 (1997)
4. Chaudhuri, S.: Query Optimizers: Time to Rethink the Contract? In: *SIGMOD 2009: Proceedings of the 35th SIGMOD international conference on Management of Data*, pp. 961–968. ACM, New York (2009)
5. Chaudhuri, S., Narasayya, V., Ramamurthy, R.: A Pay-As-You-Go Framework for Query Execution Feedback. *Proc. VLDB Endow.* 1(1), 1141–1152 (2008)
6. Cole, R.L., Graefe, G.: Optimization of Dynamic Query Evaluation Plans. In: Snodgrass, R.T., Winslett, M. (eds.) *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data*, Minneapolis, Minnesota, May 24-27, pp. 150–160. ACM Press, New York (1994)
7. Eckerson, W.W.: *Performance Dashboards: Measuring, Monitoring, and Managing Your Business*. John Wiley & Sons, Chichester (2006)
8. Evrendilek, C., Dogac, A., Nural, S., Ozcan, F.: Multidatabase Query Optimization. *Distrib. Parallel Databases* 5(1), 77–114 (1997)
9. Fagin, R.: Combining Fuzzy Information from Multiple Systems. *J. Comput. Syst. Sci.* 58(1), 83–99 (1999)
10. Goodenough, J.B.: Exception Handling: Issues and a Proposed Notation. *Commun. ACM* 18(12), 683–696 (1975)
11. Gounaris, A., Paton, N.W., Fernandes, A.A.A., Sakellariou, R.: Self-Monitoring Query Execution for Adaptive Query Processing. *Data Knowl. Eng.* 51(3), 325–348 (2004)
12. Graefe, G.: Query Evaluation Techniques for Large Databases. *ACM Comput. Surv.* 25(2), 73–169 (1993)
13. Graefe, G.: Iterators, Schedulers, and Distributed-Memory Parallelism. *Softw. Pract. Exper.* 26(4), 427–452 (1996)
14. Grossniklaus, M., Norrie, M.C.: An Object-Oriented Version Model for Context-Aware Data Management. In: Benattallah, B., Casati, F., Georgakopoulos, D., Bartolini, C., Sadiq, W., Godart, C. (eds.) *WISE 2007*. LNCS, vol. 4831, pp. 398–409. Springer, Heidelberg (2007)
15. Ives, Z.G., Florescu, D., Friedman, M., Levy, A., Weld, D.S.: An Adaptive Query Execution System for Data Integration. *SIGMOD Rec.* 28(2), 299–310 (1999)
16. Kabra, N., DeWitt, D.J.: Efficient Mid-Query Re-Optimization of Sub-Optimal Query Execution Plans. *SIGMOD Rec.* 27(2), 106–117 (1998)
17. Kossmann, D.: The State of the Art in Distributed Query Processing. *ACM Comput. Surv.* 32(4), 422–469 (2000)
18. Manolescu, I., Bouganim, L., Fabret, F., Simon, E.: Efficient Querying of Distributed Resources in Mediator Systems. In: Meersman, R., Tari, Z., et al. (eds.) *CoopIS 2002, DOA 2002, and ODBASE 2002*. LNCS, vol. 2519, pp. 468–485. Springer, Heidelberg (2002)
19. Rao, J., Pirahesh, H., Mohan, C., Lohman, G.: Compiled Query Execution Engine Using JVM. In: *ICDE 2006: Proceedings of the 22nd International Conference on Data Engineering*, p. 23. IEEE Computer Society, Washington (2006)
20. van Reeuwijk, C.: Maestro: A Self-Organizing Peer-to-Peer Dataflow Framework Using Reinforcement Learning. In: *HPDC 2009: Proceedings of the 18th ACM International Symposium on High Performance Distributed Computing*, pp. 187–196. ACM, New York (2009)



21. Srivastava, U., Munagala, K., Widom, J., Motwani, R.: Query Optimization Over Web Services. In: Dayal, U., Whang, K.Y., Lomet, D.B., Alonso, G., Lohman, G.M., Kersten, M.L., Cha, S.K., Kim, Y.K. (eds.) VLDB, pp. 355–366. ACM, New York (2006)
22. Tanin, E., Beigel, R., Shneiderman, B.: Incremental Data Structures and Algorithms for Dynamic Query Interfaces. *SIGMOD Rec.* 25(4), 21–24 (1996)
23. Urhan, T., Franklin, M.J.: XJoin: A Reactively-Scheduled Pipelined Join Operator. *IEEE Data Eng. Bull.* 23(2), 27–33 (2000)
24. Urhan, T., Franklin, M.J., Amsaleg, L.: Cost-Based Query Scrambling for Initial Delays. *SIGMOD Rec.* 27(2), 130–141 (1998)
25. Whiting, P.G., Pascoe, R.S.V.: A History of Data-Flow Languages. *IEEE Ann. Hist. Comput.* 16(4), 38–59 (1994)
26. Wong, E., Youssefi, K.: Decomposition—A Strategy for Query Processing. *ACM Trans. Database Syst.* 1(3), 223–241 (1976)
27. Yu, J., Buyya, R.: A Taxonomy of Scientific Workflow Systems for Grid Computing. *SIGMOD Rec.* 34(3), 44–49 (2005)

# Chapter 13:

## Liquid Queries and Liquid Results in Search Computing

Alessandro Bozzon<sup>1</sup>, Marco Brambilla<sup>1</sup>, Stefano Ceri<sup>1</sup>, Piero Fraternali<sup>1</sup>,  
and Ioana Manolescu<sup>2</sup>

<sup>1</sup> Dipartimento di Elettronica e Informazione, Politecnico di Milano,  
Piazza Leonardo da Vinci 32, 20133 Milano, Italy

{alessandro.bozzon,marco.brambilla,stefano.ceri,  
piero.fraternali}@polimi.it

<sup>2</sup> INRIA, Saclay-Ile-de-France and LRI, Université de Paris Sud-11, France  
ioana.manolescu@inria.fr

**Abstract.** Liquid queries are a flexible tool for information seeking, based on the progressive exploration of the search space; they produce “fluid” results which dynamically adapt to the shape of the query, as a liquid adapts to its container. The liquid query paradigm relies on the SeCo service mart and multi-domain query execution concepts: an expert user selects a priori the service marts relevant to the information seeking task at hand and the connections necessary to join them, and publishes such a definition in the SeCo back-end. The Liquid Query client-side interface consumes the application definition created by the expert and dynamically builds a query interface for the end-user. Such interface allows one to supply keywords to query the pre-configured service marts and offers controls for exploring the combinations computed by the SeCo execution engine. The interaction commands are based on a tabular representation of results and comprise: reordering, clustering, addition or deletion of attributes, addition of extra service marts to the query for specific items in the result set or for the entire result set, request of more results from all services or from selected ones, expansion of details on selected items, and more. The Liquid Query is equipped with multiple data visualization options suited to render multi-domain results and can be instrumented with indicators showing the quality of the result set.

**Keywords:** user interfaces, exploratory search, search computing.

## 1 Introduction

As users get more and more acquainted with the use of the Web for addressing their information needs, the role played by search engines in both professional and everyday life grows. A recent study by Yahoo confirms the centrality of search in Web usage: one out of every five page views of the analyzed data set is related to some search task [21]. Initially, search engine interfaces were primarily exploited for locating specific documents. The “Google-style” user interface is the perfect example of this paradigm; it offers only essential commands: an input textbox for inserting keywords, producing a ranked result list. When users have more sophisticated

information needs, they are left with the burden of alternating the necessary queries and result look-ups for locating the content of interest.

However, finding documents is no longer the only and primary way of using the Web. The same survey on online search behavior [21] found that search sessions tend to become longer (involving 6.3 page views on average) and focus not only on documents but also on structured objects (over half of the analyzed queries referred to a well identified object, which played a central role in the information seeking behavior of the user). As the expectations of users change, the user interfaces we offer them for searching must evolve too. This evolution must take into account the expected search behavior and skills of the user, the kind of content targeted by the queries, and the context where the interaction takes place (user's intention, device, situation, and so on).

The anatomy of user's search behaviors is a well investigated research topic, starting from a seminal paper by Andrei Broder at IBM [6], who distinguished information seeking needs into three main categories:

- **Navigational:** where the intent is to reach a particular site.
- **Informational:** where the intent is to acquire information.
- **Transactional:** where the intent is to perform some web-mediated activity.

More recently, Gary Marchionini [23] analyzed the evolution of search systems for **exploratory search**, defined as the situation in which the user starts from a not-so-well-defined information need and progressively discovers more on his need and on the information available to address it, with a mix of look-up, browsing, analysis and exploration. Exploratory search is showcased by several last-generation search systems, using a variety of different tools: dynamic faceted taxonomies (as used e.g., in *DBLP Faceted Search* and *Clusty.com* [8] [9] [10][29]), topic exploration engines (e.g., the *Kosmix* topical search engine [27]), Web applications aggregating community feedback and social wisdom (e.g., as in the *Hunch* problem solving engine [16]) are only a few examples.

In parallel to the evolution of the user's behavior, search solutions have evolved also in the data they can collect and present in response to a query. In the realm of textual data, the focus has shifted from document crawling and indexing to the **integration of heterogeneous data sources**, where documents are integrated with semi-structured or structured data coming from the deep Web [3] and enriched with semantics, extracted either manually or automatically. This impacts both the way in which queries are formulated (e.g., the *Wolfram Alpha* search engine [34] accepts in input structured expressions with a domain specific syntax and semantics, like mathematical formulas and stock comparison sequences) and the way in which results are presented (e.g., in the *Google Squared*<sup>TM</sup> system [12] the user may perform a plain keyword search, and the system responds with data organized in tabular format, which can be extended both vertically, by adding further "objects", and horizontally by inspecting more attributes of the tabbed objects).

The SeCo liquid query interface sits at the intersection of the abovementioned trends. On one side, it is based on the **structured information** collected by the SeCo back-end architecture from both Web documents and deep Web data sources, wrapped by means of a uniform notion of search service. On the other, it addresses also **exploratory search**: the user may start from an initial combination of data

sources relevant to his information need and then explore the content of these services, or explore the service universe by following novel trails based on other “joinable” services.

Another influence on the design of the liquid query layer comes from the emergence of **search-based application development** as a distinctive field of online application development [4]. The functionality of a search system is unbundled into a set of reusable components, which can be integrated to assemble tailor-made search solutions (as an example, see the *Microsoft Symphony* platform [30]). Expert users, although not necessarily trained in computer programming and code-based application development, are offered sophisticated interfaces for assembling or configuring ad hoc search solutions from existing resources, possibly using a *mashup* approach (see, for example, the *Yahoo Pipes* platform which allows the *mashup* of data extracted from the Web with the *Yahoo Query Language* [36][37]). In SeCo, the “expert users” will exploit the graph of service marts and prepare queries for a specific application, thanks to a *mashup* interface over service marts and their connections; conventional “end users” will use such preconfigured queries, by supplying their actual search parameters and perusing the results with variety of commands for personalizing their search experience and data visualization.

This chapter is organized as follows. Section 2 discusses the state of the art in interfaces for Web search. Section 3 describes the approach offered to expert users for building queries while introducing the functionalities offered to end users for interacting with liquid results. Section 4 shows the liquid result interface at work on a running example; and Section 5 illustrates the current state and future work.

## 2 State of the Art

The design of the liquid query interface draws from the achievements in a number of fields related to the development of interactive systems for information seeking. In this Section we review the most prominent studies, systems and solutions that are at the base of new generation search interfaces.

### 2.1 Behavioral Studies of Information Seeking and Exploratory Search

The design of novel search systems and interfaces is backed by several studies aimed at understanding how users behave when satisfying their information needs on the Web. After the seminal work of Broder [6], other studies have investigated search behaviours by analysing search engine logs. For example, the study performed on queries selected from the *Altavista* logs by Rose and Levinson highlighted a taxonomy of search goals comprising informational, resource and navigational queries [28], with a prevalence of informational queries aimed at learning more about a topic of interest. These first studies, where queries were identified and classified manually by inspecting the logs, have been followed by several attempts to automate the classification process, to cater for larger scale inference of the intent behind user’s searches (examples are [21], [22], and [33]). A review of approaches to search log data mining and Web search investigation is contained in [2] and [18][19].

A specific class of studies is devoted to **exploratory search**, a close relative of informational queries where the user's intent is primarily to learn more on a topic of interest [23]. Such information seeking behaviour challenges the search engine interface, because it requires support to all the stages of information acquisition, from the initial formulation of the area of interest, to the discovery of the most relevant and authoritative sources, to the establishment of relationships among the relevant information elements.

A good definition and analysis of the problem are given in [33] and an interesting distinction between complex and exploratory search is made in [1], where complex search is characterized by:

- multiple searches, possible over multiple sessions and spanning multiple sources of information,
- combination of exploration and more directed information finding activities,
- need of note-taking,
- variation of the search goal during the search process.

A number of techniques (some of which are reviewed in Section 2.2) have been proposed to support exploratory search, and user studies have been conducted to understand the effectiveness of the various approaches (e.g., [20]).

Besides field studies, a model-theoretic approach to the analysis of the information seeking behaviour is afforded by the theory of Information Foraging [26], which applies evolutionary ecological models of foraging to knowledge acquisition tasks. The theory relies on *information patch models*, which explain how a user decides between moving from an information patch (e.g., a search engine result list or a topic page) to another one or stopping to exploit the content of a patch (e.g., navigating to an item in the result list or exploring a topic); and on *information diet models*, which convey the policy used by users to select a profitable mix of heterogeneous information sources. The theory has been embodied in a production system, which simulates the information foraging strategies for a knowledge acquisition task and derives predictions of the actual decisions occurring in real tasks observed during field studies. It also provided practical hints on how to make the interface for accessing information more efficient, e.g., by enriching the productivity of information patches by means of filters that eliminate non-profitable information, or by strengthening the *information scent*, i.e., the clues that the user exploits to decide if an item is worth exploring.

## 2.2 Topic Exploration Systems

Topic exploration is a case of complex and exploratory search, centred around the goal of collecting information on a subject matter of interest, from multiple sources. The key challenge in topic exploration on the Web is the massive amount of disparate information available on each topic, which demands novel systems capable of constructing effective entry points for quickly grasping the essence of a topic and the possible directions for its exploration.

Topic exploration has been traditionally served by vertical search engines (e.g., *WebMD*, *Mobissimo*, *Google News*, *CareerBuilder*, *MP3.com*), which restrict

the scope of the available topics to a specific domain. Horizontal, i.e., cross-domain, topic exploration is a recent development.

### 2.2.1 Kosmix

**Kosmix** [27] is a general-purpose topic discovery engine, which responds to keyword search by means of a **topic page** that summarizes the most relevant information on the subject associated to the search.

Each topic page is constructed by evaluating a set of modules, which are software components that wrap calls to Web services to extract information from deep Web data sources. Topic pages are defined manually and may contain different modules: e.g., images from *Flickr* and *Google*, topic descriptions from *Wikipedia*, reviews and guides from domain-specific data sources, news from magazines and aggregators, product offerings from *eBay* or *Amazon*, and more.

Internally, Kosmix uses a mix of crawling and federated search: part of the data are crawled and indexed statically, part are fetched by calls to external web services at query time. Query processing exploits a taxonomy of topics, comprising millions of nodes connected in a direct acyclic graph, and a Categorization Service, which computes the nodes of the taxonomy that are most closely related to the user's query and the data sources in the system that can provide information about the query topic.

When the relevant sources of information have been identified, the Kosmix engine performs the necessary data source queries and assembles the result in the topic page, which has a bi-dimensional layout similar to that of a magazine (see Fig. 1 for the topic page associated to the “Leonardo da Vinci” keyword search).

The topic page may also contain a “Related in the Kosmos” module (see Fig. 2), which highlights the related topics found in the taxonomy, grouped by categories.



**Fig. 1.** The Kosmix topic page for the “Leonardo da Vinci” keyword search. The system proposes a summary page.

**Related in the Kosmos** ?

**Leonardo da Vinci paintings**

- [The Virgin of the Rocks](#)
- [The Mona Lisa](#)
- [The Last Supper \(Leonardo\)](#)

**Tuscan painters**

- [Leonardo da Vinci \(personal life\)](#)
- [Verrocchio](#)
- [Vasari](#)

**American novelists**

- [Clancy, Tom](#)
- [Kate Chopin](#)
- [Dan brown](#)

**Works by Leonardo da Vinci**

- [The vitruvian man](#)
- [Leonardo da Vinci paintings](#)

**Italian civil engineers**

- [Vitruvius](#)
- [Leonardo da Vinci \(personal life\)](#)

**See also** (20)

- [Michael J. Gelb](#)
- [The Priory of Sion](#)
- [Vinci, Italy](#)
- [Leonardo Leonardo](#)
- [The Gospel of Philip](#)

[+ MORE](#)

**Fig. 2.** The “Related in the Kosmos” module of the topic page for “Leonardo da Vinci”

Factz from Wikipedia: we found the following about Leonardo da Vinci

<a href="#">Leonardo da Vinci</a>	painted :	Mona Lisa, portrait, Last Supper, Battle, Lady, La Gioconda, supper, Bacchus, delle, style, copies, <a href="#">more</a>
<a href="#">Leonardo da Vinci</a>	designed :	automaton, Horn Bridge, <a href="#">aircraft</a> Horse, staircase, system, proto-tank, navigli, Valle Camonica, <a href="#">more</a>

Results for “Leonardo da Vinci designed aircraft”

[Fixed-wing aircraft](#) [Leonardo da Vinci](#) researched the wing design of birds and [designed](#) a man-powered [aircraft](#) in his Codex on the Flight of Birds (1502).

invented :	machine, flowers, automata, soldier, press, database, boat, Traghetto di Leonardo, Renaissance and wheellock.
------------	---

[more](#) showing 3 of 85

**Fig. 3.** The facts extracted from *Wikipedia* about Leonardo da Vinci; each fact (e.g., “Leonardo designed aircraft”) is supported by a reference to the information source)

### 2.2.2 Other Topic-Based Systems

The organization of topical information is the goal of a variety of systems that employ different approaches and technologies to collect and layout the relevant information on a subject matter related to the user’s search.

**Powerset** (now incorporated into Microsoft’s Bing [24]) specializes in extracting and organizing information from *Wikipedia*. A summary page is produced for each topic associated to a keyword search, which contains the essential facts and articles (e.g., biographical data for an historical figure). *Wikipedia* articles are summarized and augmented with reading aids (e.g., an outline browsing panel) and facts are extracted from them (e.g., the facts discovered about “Leonardo da Vinci” in Fig. 3).

**Hakia** [15] is another search engine capable of producing summary pages for topics associated with user’s queries. Hakia exploits natural language processing techniques, specifically Ontological Semantics, for building a large ontology of concepts and correlations and for parsing text content into an ontological representation. Web pages are indexed with the Query Detection and Extraction (QDEX) System, which analyses the page content in order to determine all the possible queries that can be responded with that content. Such meaningful queries are represented internally by means of a concise ontological model, which replaces the

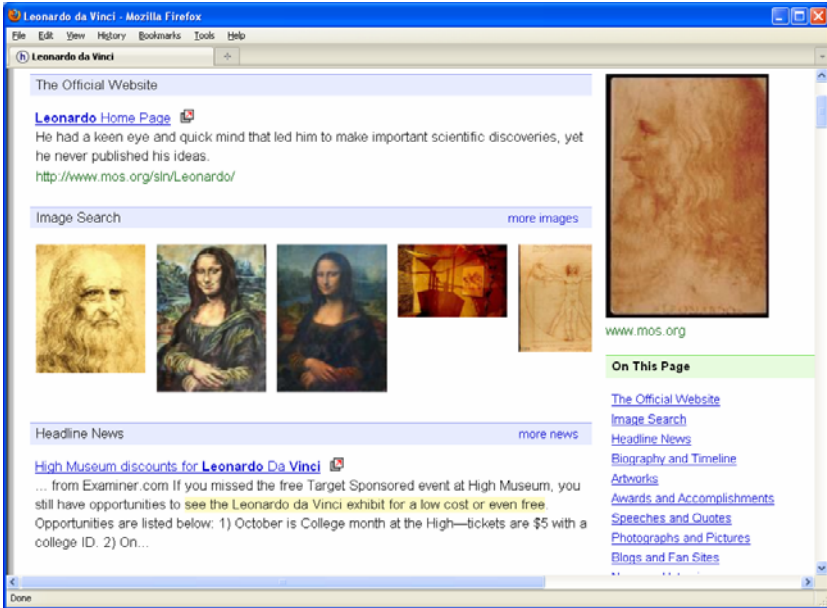


Fig. 4. The resume page for the query “Leonardo da Vinci”

standard inverted index of a classic text-based search engine. QDEX data for a given query are then used to rank the query results, by semantically matching the query terms and the QDEX sentences, so to determine the index entries most meaningful for answering to the query. A topical query is answered by means of resume page, which organizes the relevant pieces of content categorically, as shown in Fig. 4.

**Parallax** [14] is an interface for browsing **Freebase**, a large collaborative knowledge base, where structured, linked data are harvested from several sources, including *Wikipedia*, *ChefMoz*, *NNDB*, and *MusicBrainz*, and enriched with user generated content. Freebase data are organized into topics and stored according to ontologies that can be updated by users. Parallax queries are sets of keywords, which are disambiguated in order to identify the relevant topics. Topic information is presented to the user and faceted navigation can be used to move from the current data set to a related data set, exploiting the Freebase connections. Moreover, Parallax enables the navigation along topics, leveraging the semantic associations recorded in the Freebase ontologies.

### 2.2.3 Hybrid Search

A somehow hybrid position between vertical search engines and topic exploration systems is taken also by the latest versions of the mainstream, general-purpose search engines interfaces, which are enriching results lists with extra elements derived from vertical or topical searches.

Examples of these extensions are *Google Universal Search*, *Ask 3D* and *Microsoft Live Search*. For example, Fig. 5 shows the results of the keyword search “Leonardo da Vinci” in *Ask*.





**Fig. 5.** Ask 3D search result page for the query “Leonardo da Vinci”. The page mixes results from traditional horizontal search and from vertical searches in news, blogs, topical Web sites, image repositories, and more.

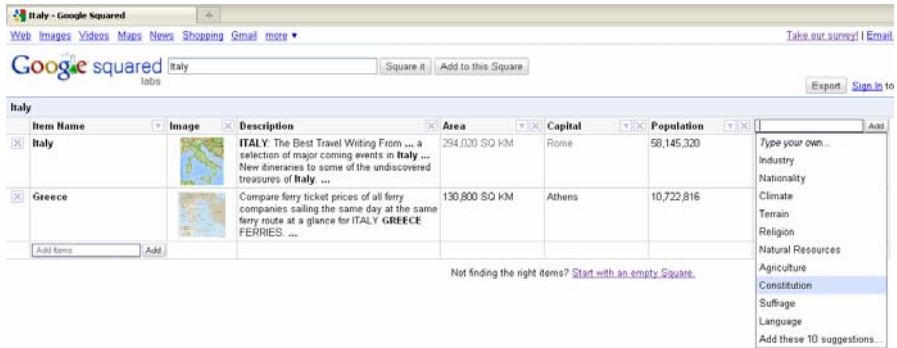
**Yahoo SearchMonkey** [35] brings result page enhancement into the hands of developers, by allowing them to add their own structured data to Yahoo result lists, to make them appear more informative. Extensions can be defined either by completely rewriting the standard result list or by adding information bars to result elements.

### 2.3 Tabular Search Systems

Recently, a number of experimental systems have been investigated with the purpose of merging the popular keyword search paradigm with the tabular representation typical of structured data in such applications as information systems, relational data interfaces, spreadsheets, and data warehouses. The novelty of these approaches resides in the capability to extract approximate schema information directly from web documents and the idea of enriching structured data (e.g., spreadsheets contributed by the user) with related data mined from the web.

#### 2.3.1 Google Squared

Google Squared [12] is an application from Google Labs demonstrating the interplay between Web data and schemas overlaid on top of it [7]. Fig. 6 depicts a screenshot of the Google Squared interface: the interaction can be started by an ordinary keyword



**Fig. 6.** The interface of Google Squared. A square has been constructed starting from the keyword search for “Italy” and manually extended with the term “Greece”. The system suggests correlated columns to expand the square horizontally.

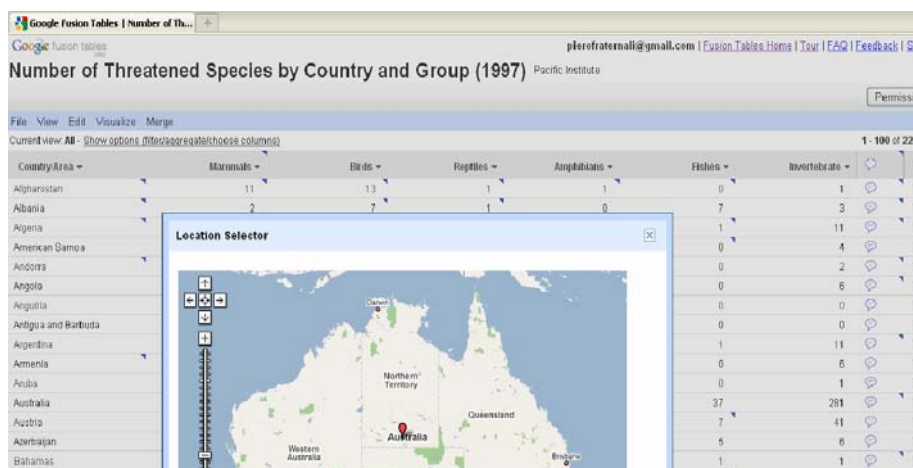
search, but the results are collected in a table (called a *square*) featuring all the attributes relevant to the result items as columns headers. The initial square can be extended horizontally, by adding columns suggested by the system or by providing tentative column names. If a new column is added to the square, the system tries to locate data matching the supposed semantics of the column and extends the square with the retrieved data. Similarly, the square can be extended vertically; the user can provide new items of the same type of those already listed in the table, or the system can provide suggestions of new items that have attribute values similar to those of the items already listed in the square.

### 2.3.2 Fusion Tables

Google Fusion Table [11] is an application developed at Google labs. The interface, shown in Fig. 7, allows one to upload a data table (e.g., a spreadsheet file) and join (or “fuse”) the data in some column with other tables, either supplied by other uses or mined from the Web. Spreadsheet-like views of the base or joined tables can be defined, saved, shared with others, and commented collaboratively. Alternative visualizations are possible, depending of the type of data contained in a table, e.g., timelines and maps.

## 2.4 Summary and Discussion

The Liquid Query system can be considered as an interface for the exploration of ranked combinations of objects, extracted from deep Web data sources. As such, it shares aspects with exploratory search solutions and with tabular search systems. The nature of the result set (combinations of objects with given properties) suggests the use of the tabular format as the primary, but not the unique, result presentation metaphor. However, differently from *Google Squared* and *Fusion Tables*, the presence of service mart signatures allows recognizing the boundaries of objects of different types and thus structuring each row in the result set into the individual objects used to build it.



**Fig. 7.** The interface of *Google Fusion Table*, showing the table of endangered species by country and group. Each column can be annotated and metadata can be displayed (e.g., the map location of a country)

As in the case of topical search engines, the approach to the exploration of the search space is structured in two steps: a *configuration* phase and a *usage* phase. In the configuration phase (discussed in Chapter 14), an expert user defines the universe of exploration, by selecting the service marts relevant to the envisioned information seeking task and their connections; this is analogous to the offline preparation of a topic page, in which the appropriate data sources for the topic are preselected. Differently from topical search, which focuses on a single type of object, on its properties, and on one step of relationships to other objects, the Liquid Query interface addresses the combinations of objects of different types, with their attributes; each object type in the combination is the source of further exploration steps, represented by multi-hop outgoing paths along service mart connections preselected by the expert user.

In the usage phase, the end-user exploits a form based interface to provide the constants of his specific query over the preselected service marts. The retrieved top-k combinations are then displayed and make the initial scenario for exploration. The user can filter and reshape the result set, explore the vicinity of the search space by following new trails from the currently visualized items, or “change the information patch”, by asking for further results of the same query from one or more services (which may alter the combinations appearing in the top-k list) or posing a novel query with different parameters.

### 3 Liquid Query Paradigm

The Liquid Query interface offers a set of interaction options to ease information exploration by end users. In this section we describe the essential interface concepts, the flow of the user interaction, and the main primitives composing such flow.

### 3.1 Liquid Query Concepts

The main objects involved in the query lifecycle are:

- **Liquid Query:** a concrete query upon service interfaces specified by inputting the constant terms to use in the selection operators, equivalent to the queries formally introduced in Chapter 10;
- **Liquid Result:** a list of tuples, representing object combinations, conforming to a **liquid result schema**; the liquid result schema is established a priori by the expert user at application design time, but can be changed by the end user during interaction, for example by projecting away attributes or extending it with additional service interfaces not present in the schema of the initial query.
- **Interaction Primitive:** a specific user command that produces a side effect either on the liquid result (e.g. grouping by given attributes, selecting or re-ordering tuples) or on the liquid query (e.g. requesting more tuples, or expanding the result by joining in new service interfaces).

The **liquid result schema** defines the format of the result set, in terms of displayed attributes, ordering attributes, clustering attributes, grouping attributes, and available expansions. A **liquid result instance** is a tuple that represents a combination of objects extracted by the SeCo engine in response to the query, graphically shown as a row in the tabular representation of the liquid result set. A **liquid result page** is a set of liquid result instances displayed simultaneously in the user interface. The number of instances per page is the same as the number  $k$  of results returned by the SeCo engine in the computation of the top- $k$  combinations, which is pre-configurable by the expert user at application development time.

A **query expansion** is an operation that allows users to select some tuples in the result set and join them with objects provided by another service interface, called the *expansion target*. The join operation exploits pre-selected connections, taken from the service mart repository; and the join attributes of the selected tuples provide the values necessary to execute the join operation required for the expansion. Visually, the objects retrieved from the expansion target service interface are joined with the tuples that provided input values for their extraction, and displayed alongside the results of the query; in this way, a new service is dynamically added to the liquid result. The tuples originated from an expansion can be further used for more expansions, allowing a stepwise exploration of the search space vicinity of the initial result set.

### 3.2 Liquid Query Interaction Process

The first step of the user interaction consists in the **query submission**, whereby the end user specifies the actual parameters of the liquid query. The query is created by an expert user or application developer by using the available service interfaces and connection patterns (see Chapter 14 for further details).

The **query execution** step produces on the server side a result set of  $k$  tuples, which satisfy the query predicates and are ordered. Depending on the query and execution plan, the  $k$  result tuples are either guaranteed to be exactly the top- $k$  tuples, ordered according to the ranking function, or instead they are  $k$  tuples extracted by an

approximation of that ranking; the latter method is faster. The result set is then passed to the client, where it is displayed in a personalized manner according to the user's visualization choices (e.g., attribute projection, sorting, grouping, etc). More precisely, the following actions can be locally performed on the result set stored on the client side:

- The client-side result set can be restricted by applying local selection conditions, if the user has set local filters to the query result;
- Instances can be grouped, if the end user has defined a grouping attribute, and sorted with respect to the chosen ordering attributes, if these have been provided by the user. A local ordering specification overrides the ranking function used at the server side for computing the top-k results;
- The result set can be expanded, by invoking the SeCo back-end, and by placing the expansion result visually to the right of the service results that used as input values for the expansion;
- Finally, items can be clustered according to the clustering attributes, if these have been specified by the user.

Results are then displayed to the user, who can then start the **result browsing and query refinement** phase, in which the user can examine and manipulate the results through appropriate interaction primitives, which update either the liquid result or the refine the liquid query. When browsing the result set, the interaction primitives may access the server-side (*Remote Query Interaction Primitives*), or affect only the client-side result set (*Local Result Interaction Primitives*). Depending on the kind of remote query interaction primitive, the query execution performed by the SeCo engine might be suspended, restarted, or stopped.

Besides the basic query and result interactions, we envision other classes of interactions: *Manipulation Primitives* for defining calculated data; *Visualization Primitives* for changing the result set visualization; *Query Management Primitives* for storing/retrieving, exporting, and sharing queries; and *Result Quality Primitives*, for understanding quality parameters of the result set, such as relevance and diversity, and for capturing user's preferences. In the following, we describe in detail the Remote and Local query interaction primitives, and preview the other classes of interactions, which are part of our future work.

### 3.3 Remote Query Interaction Primitives

Remote interaction primitives require the client to ask the server for some computation, in order to produce new results. Remote interaction primitives include:

- **MoreAll:** the operation loads additional tuples from all the selected services in the currently specified query (excluding extensions); this command is typically executed when the user has not found the combination he is looking for in the top-k results and cannot estimate the service more likely to provide profitable information. This operation increases the cardinality of the result set, without altering the ranking function associated with the query.
- **MoreOne (Service):** the operation asks for additional tuples from a specific service interface in the currently defined query. This command is typically

executed when the user has not found what he is looking for in the top-k results and can identify the service that returned unsatisfactory information. Notice that this operation does not preserve the ranking of the result set as computed according the ranking function of the query, because new objects (and thus new combinations) may be computed for a single service, which does not guarantee to form the best combinations possible.

- **Expand (Target Service, Selected Tuples):** the operation expands the result schema by adding one new target service and joining it with selected tuples. The expansion causes a set of exact queries to the expanded service interface, with input derived from the join attributes of the tuples selected by the user. If the expansion target service interface requires additional inputs, a dialog box is shown to the user for submitting the needed parameters.
- **ChangeRank (Weights):** the operation modifies the ranking function by updating one or more weights of the linear combination. This operation is performed upon results that are cached on the server, without re-computing the query, but causing their reload on the client in a different order. Notice that if the query uses the FA algorithm yielding top-k optimal results (discussed in Chapter 11) then a change rank operation still produces the top-k results, because the method does not depend on the choice of the rank function.

### 3.4 Local Query Interaction Primitives

Local interaction primitives permit the user to personalize the presentation of the result set cached at the client side, without requiring the invocation of the server tier of the SeCo architecture. They comprise:

- **Group (Attribute):** the operation collects results having common values for the specified attribute in a group. The group assumes as a title the attribute value at hand. Notice that this operation can be performed only on one attribute at time. By applying this operation, all the clustering and sorting options possibly defined by the user are applied separately to each group.
- **Ungroup:** the operation removes the existing grouping option.
- **Cluster (Attribute/Service):** the operation changes the visual appearance of the result list by clustering adjacent tuples on a specific attribute and hiding duplicate values. Notice that sorting and grouping are not modified. If clustering is defined on multiple columns, it will be actually applied following the horizontal order of the columns, i.e., leftmost columns will be clustered first. The operation can also be applied to all the columns of a service interface in one shot.
- **Uncluster (Attribute/Service):** the operation undoes any preceding cluster operation at attribute or service level; sorting and grouping are not modified.
- **Sort (Attribute):** the operation sorts the currently displayed results w.r.t. the values of an attribute. Notice that the clustering definition is not modified. However, clusters may recombine because instances that were adjacent in the previous order may no longer be contiguous, and vice versa. Notice that if grouping is applied, ordering is applied on items within each group.
- **Unsort (Attribute):** the operation undoes a sort operation.

- **Roll-up (Attribute):** the operation hides a currently visible attribute from the result schema. If the attribute removal introduces duplicated elements in the result list, duplicates are eliminated too. Notice that if the attribute was used for ordering, grouping, or clustering, it is removed also from the respective criteria.
- **Drill-down (Attribute):** the operation adds a new service interface attribute to the results, taken from the list of available attributes not yet displayed. Notice that new instances (rows) could appear in the result list, due to the splitting of elements that previously appeared as duplicates.
- **Filter (Condition):** the operation reduces the number of instances shown in the result list by locally apply a filtering condition on one of the displayed attribute.
- **RemoveFilter (Condition):** the operation undoes a preceding filter operation.
- **DeleteInstance (Tuple):** the operation locally deletes one instance from the currently displayed items, thus reducing the population of the current result list. This can be seen as a particular case of filter operation, based on the condition:  $\text{TupleID} \neq \text{SelectedID}$ .
- **Pivoting (MultivaluedAttribute):** a multi-valued attribute or repeating group is selected, and then all instances with the same attribute value or repeating group value are clustered together, thereby rendering the other attributes as repeating groups. Notice that pivoting is disruptive with respect to the result schema and therefore resets all the settings specified by the user up to that moment.
- **ChangeProvider (ServiceInterface, ServiceImplementation):** the provider of a specific service interface is changed; the new service interface must have exactly the same access pattern as the old one. This feature allows the user changing some qualitative aspects of the objects forming the result set, e.g., switching from a service providing standard hotels with one offering family style hotels.

### 3.5 Local Manipulation Primitives

Local manipulation primitives include a set of options for applying calculations on the data, to enrich the interpretation of the information by the user. The applicable operators are computed at client side and depend on the type of the attribute:

- **Math (Attributes):** a new attribute can be derived by applying an arithmetic expression (with the usual operators) upon numeric attributes.
- **Temporal (Attributes):** For date/time attributes, allowed operations are only Subtraction, Maximum, and Minimum.
- **String (Attributes):** For text attributes, only concatenation (+) is allowed.

The calculated attributes can be used for sorting, grouping, clustering, and join operations.

### 3.6 Data Visualization Primitives

Data visualization primitives support different visual representations of the extracted data, giving a more immediate intuition of the results. In the future, we will study how the multi-domain paradigm can benefit from existing data visualization techniques and, vice versa, how the multi-domain structure of the results and the exploratory

approach can impact on data visualization. Examples of useful data visualization primitives include:

- **Value Bar/Pie Chart (Numeric Attribute):** the user can select one (numeric) attribute and get a bar/pie chart of the results. Bars and slices become browsable objects, e.g. for navigate to instance details.
- **Aggregate Bar/Pie Chart (Attribute):** the user can get a bar/pie chart of the distribution of results over attribute values. Bars and slices become browsable objects, e.g. for navigate to instance details.
- **Correlation (Attribute1, Attribute2):** the user can select two attributes (possibly from different service results) and get a 2D X-Y graph representation of the positions of the results.
- **GeoMap (Geo Attribute(s)):** the user can select one or more geo-location attributes and get a map with the locations of all the instances. When selecting one pinpoint, he can navigate to the respective instance details.
- **Tag Cloud (Services):** various commands of this kind generate visual clouds of the concepts available in the result set, with an indication of the respective weights and relationships. Clicking on a concept restricts the result set to the instances that relate to that concept. The cloud can be built on one or more services.
- **Parallel Coordinates (Attributes):** by selecting a set of attributes (with numeric or finite domain), the user can see a set of tuples in a parallel coordinates diagram [17], that allows him to have a quick overview of the set, and to easily select a subset of instances for further exploration.

### 3.7 Query Management Primitives

The *search as a process* approach will be afforded by the Liquid Query system by enabling long-lived search session. To do so, the user must be able to manage his queries and result sets, suspending and resuming the search process. A search process can also be turned into a notification system, to alert the user when new relevant data arrive at a data source. The following primitives will provide these functionalities:

- **Export** of the current dataset in various formats (textual, spreadsheet, XML, HTML, PDF, RTF);
- **Save** the current query status; this command saves not only the data retrieved by the query but also the personalization applied to the result schema.
- **Open** a previously saved query;
- Define a **public permanent link** to the current result set view, that can be emailed, linked from web sites, or shared with friends;
- Define an **RSS/ATOM syndication** feed on the query, which informs the user when new results are available;
- Store the query as preferred bookmark on **social bookmarking systems** (*Delicious*, *Digg*, and others);
- **Navigate the query history** through the buttons *Previous*, *Next*, *First*, and *Last*, which allow the user to rollback and/or repeat the interaction history. These features will be based on application state modeling, which grants correct application behavior with respect to the navigation history even in case of heavy involvement of client-side scripting [5].



### 3.8 Result Quality Primitives

The Liquid Query interface will also be used to experiment with different heuristics for enhancing the quality of the result set as perceived by the user. In top-k search systems the quality of the result set is determined by a trade-off between relevance, which expresses how well a combination matches the user query, and other quality factors. Among these, diversity has been studied extensively [25], as a means of introducing variety into the result set and make it more attractive. For instance, given a query that comprises hotels, relevance can be measured by the parameters explicitly provided in the query or ranking functions (e.g., number of stars or distance from a location), whereas diversity could be introduced by considering hotels of different classes (design, family-stay, etc). Diversity normally clashes with relevance, because making the result set more diverse may exclude some highly relevant combination. The Liquid Query interface will incorporate suitable commands for tuning the trade-off between relevance and diversity, like:

- **Edit Object Diversity Metric:** a command for defining a function over the attributes of the result instances so to quantify the diversity between two instances (e.g., quantifying the diversity of hotel types).
- **Edit Combination Diversity Metric:** a command for quantifying the diversity of two objects combinations; the measure may be purely set-theoretic (e.g., a Jaccard measure based on the number of objects in common) or take into account the diversity metrics defined on objects (e.g., the diversity between two combinations is a function of the diversity of the constituent objects).
- **Set Relevance-Diversity Trade-Off:** the command lets the user regulate the amount of relevance he is willing to give up to obtain a more diversified result set. This could be done in several way, e.g., by specifying an absolute or percentage loss of combination score, limiting the minimum relevance score of the instances in the result set, and so on.

## 4 Running Example

This section describes a typical user interaction scenario based on the running example presented in the previous chapters; we assume that suitable search services concerning Movies, Theatres, Restaurants, and Subway Stations have been registered and that the SeCo application has been already configured properly for an end user wishing to go out for a movie and dinner. In this setting, we describe a user search interaction comprising the submission of the initial query concerning movies, theatres, and restaurants, and the refinement and exploration of results through application of additional local filters, expansion of the query, calculation of some derived information, as well as selection of clustering and visualization options for improving the readability of results. The steps are described in detail through mockups in the following subsections.

### 4.1 Initial Query Submission and Result Visualization

The Liquid Query client application exploits the application configuration created by the expert user to build the query submission interface shown in Fig. 8, comprising the input parameters needed to execute the query (e.g., *action* movies whose US opening is after

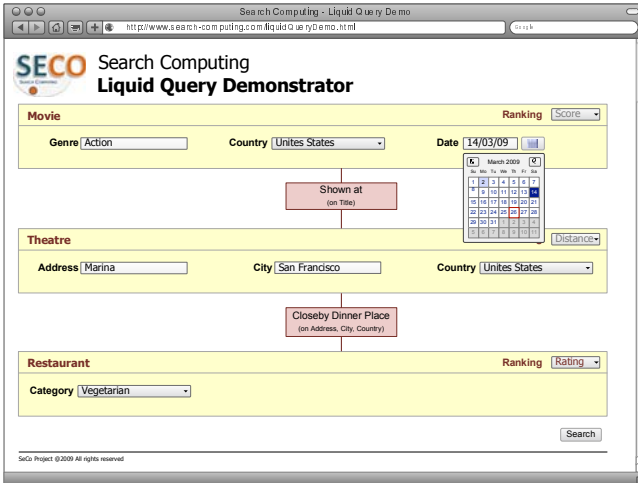


Fig. 8. Mockup of the initial query submission

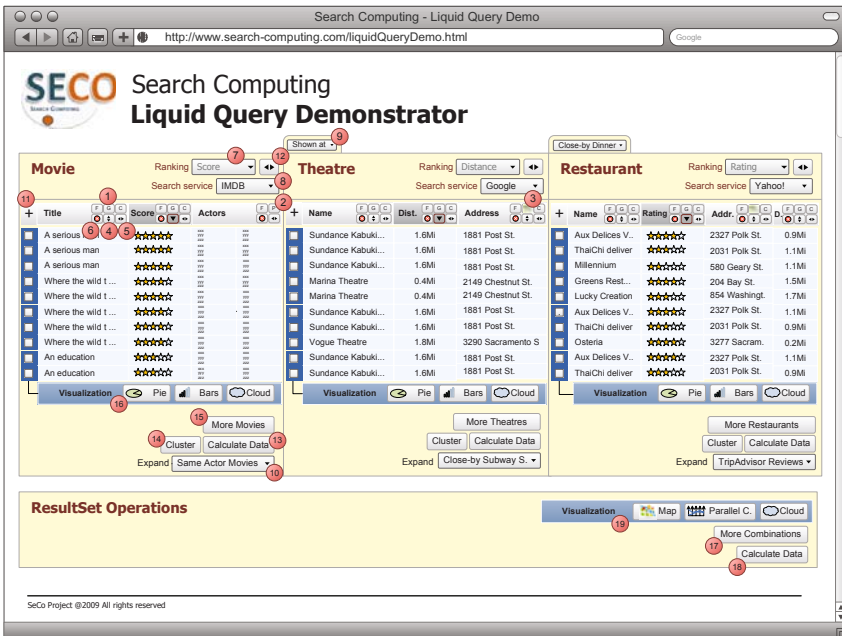


Fig. 9. Mockup of the liquid result interface

March 14, 2009, theatres close to the San Francisco Marina, and vegetarian restaurants close to the theatre). Notice that the user interface highlights the existence of service marts and of connection patterns, thus making the user aware of the searched data sources.

Once the user submits the search parameters, the query is performed and results are calculated and displayed in the result table, as illustrated in Fig. 9. The result page is enriched with the liquid interaction options that the user can choose.

**Table 1.** Summary of Liquid Query primitives and respective visual representations









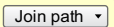















Level	Symbol	Description
<b>Attribute level</b>		
		Filter results with condition on this attribute
		Hide this attribute
		Group by this attribute
		Cluster results on this attribute
		Order by (asc/desc/none) and ordered attr.
		Move attribute position in the table
		Pivot on this multi-value attribute
		Show this attribute on a map
<b>Service level</b>		
		Change join path between services
	<b>Ranking</b> 	Change ranking attribute
		Move service position in the table
	<b>Provider</b> 	Change result provider for this service
		Get more results from this service
		Cluster results on all the attributes
		Add a calculated column from existing ones
	<b>Expand</b> 	Expand results to other services
	  	Data visualization options
<b>Resultset level</b>		
	  	Data visualization
		Get more results from all the services
		Add a calculated column from existing ones

Table 1 presents a summary of the main primitives available at the various levels (that are also highlighted by numbered dots in Fig. 9):

- At column level**, a set of buttons is shown on the column header for performing operations on the respective attribute. The available buttons depend on the type of the column: “F”, “G”, “C” buttons (1) respectively apply filters, grouping, and clustering on that attribute; “P” (2) apply pivoting to the corresponding multi-valued attribute; the “Map” symbol (3) shows column of a geo-referenced type on a map; the sorting button (represented by two vertical arrows) (4) changes the sort status of the column (Ascending, Descending, None); the move button (represented by two horizontal arrows) (5) moves horizontally the entire column within the boundary of the service; the “X” button (6) performs a roll-up on that attribute (i.e., hides the respective column) and removes duplicates.
- At service level**, the available operations are displayed as a set of dropdown lists for changing the rank attribute (7), changing the search results provider (8), changing the connection path that joins the services (9), expanding with a new

service (10); the “+” button for applying a drill-down on hidden attributes (11); the move button (represented by two horizontal arrows) (12) moves the entire column set of the service horizontally; the “Calculate Data” button (13) creates new columns starting from available ones; the “Cluster” button (14) applies clustering to all the columns of the service; the “More” button (15) asks for more results from the specific service; and a set of visualization buttons (Pie, Bar, Cloud) (16) show the service results with different renditions.

- **At combination level**, the “Calculate Data” button (17) creates calculated columns from the ones available within the whole combination (the new column will not belong to any service); the “More” button (18) asks for more result instances; visualization buttons (Map, Parallel Coordinates, Cloud) (19) show the combinations in the result set in different ways.

Once the results are shown, the user can interact with them through the available commands. Some operations (i.e., visualization options and expansion to new services) require the user to select a subset of result instances; selection is performed by means of checkboxes. When needed, a popup window asks for additional parameters or details on the operation to be performed.

### 4.2 Application of Local Filters

Let’s suppose now that the user wants to select only the restaurants having rating higher than three stars. Local filters on column values can be applied by clicking the “F” button on the column header of interest. The button triggers a dialog window (Filter results) with a simple interface for editing conditions, as shown in Fig. 10. The form supports Boolean expressions of simple predicates.

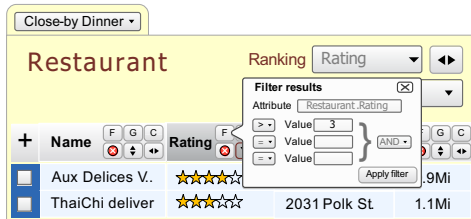


Fig. 10. Mockup of the result filtering form

### 4.3 Query Expansion

Subsequently, if the user is interested in the list of subway stations close to the listed theatres, he can select a subset of theatres of interest and ask for the needed expansion from the dropdown list. This produces a new service result, with the values of the subway stations for the selected theaters, as shown in Fig. 11. In the example, we suppose that the new information is produced by invoking the *BART* Web Service.

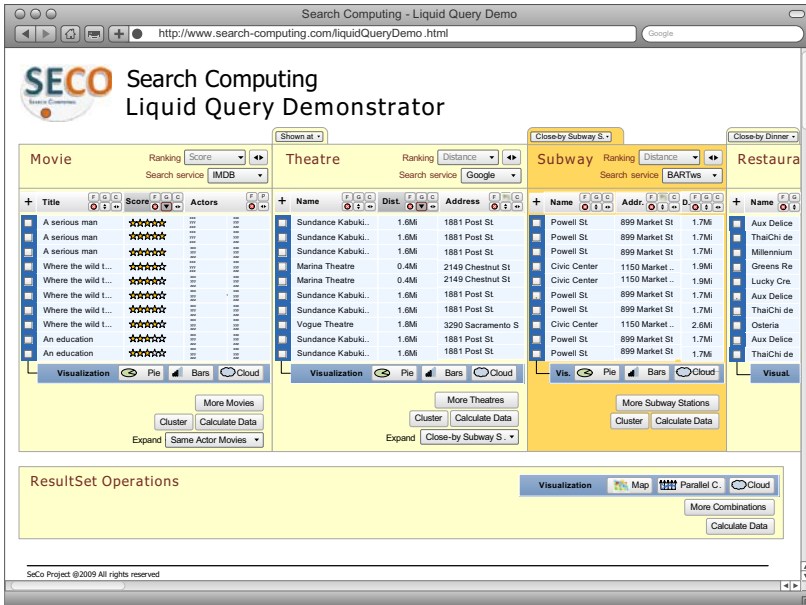


Fig. 11. Mockup of the liquid result expanded with the Subway Stations information

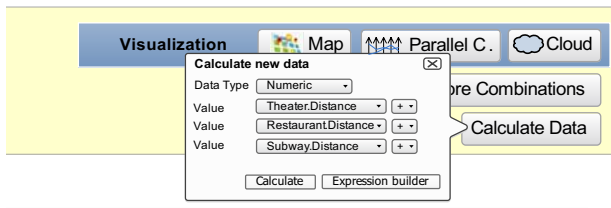


Fig. 12. Mockup of the calculate data popup window

#### 4.4 Adding Calculated Attributes

Let's suppose the user wants to calculate the total walking distance for the planned night. This will consist in a simple sum of the distances between the locations mentioned in the result set. The user can add new calculated columns at service level (i.e., only involving attributes from a single service) or at combination level. Fig. 12 shows a popup window for defining a new calculated column at combination level. The user objective can be reached simply by selecting the attributes in the form, together with the *sum* operator (+). If more sophisticated calculations are needed, the user can click on the Expression builder link and be redirected to an appropriate expression editor. Since the new calculated data is at combination level, the corresponding column will appear as the last one in the table and will not belong to any services.

#### 4.5 Visualization of the Results on a Map and on Parallel Coordinates

Finally, the user may want to change the visualization options for the results. For instance, all the geo-referenced values of the result set can be visualized in a Map. The effect is shown in Fig. 13. In the example, three attributes of type address were identified (address of theatres, address of restaurants, and address of subway stations) and positioned in the map together with a legend.

The user can also select an alternative visualization option, e.g., parallel coordinates [17] (see Fig. 14), in which all the instances in the result set are organized by different dimensions (e.g., score of the movie, distance of the theatre, duration of the movie, distance and score of the restaurant). The diagram allows the user to interact with the results, by graphically selecting a dimension and restricting the associated values. For instance, if the user selects the distance of the restaurant between 1.1 and 1.75 miles, the corresponding results will be highlighted in the graph. Selecting one result instance displays the detailed information about the chosen combination.

#### 4.6 Query Management Operations

The user can manage the query and the result set through the Query Management panel, that allows him to export the current dataset in various formats, store and reload the current query status, define a public permanent link, define an RSS/ATOM syndication feed, store the query as preferred link on social bookmarking systems (*Delicious* and others), and so on. Moreover the user can navigate the query and browsing history through the query history navigator panel. Fig. 15 shows the mockup of the panel that the user can open at any time during his search task.

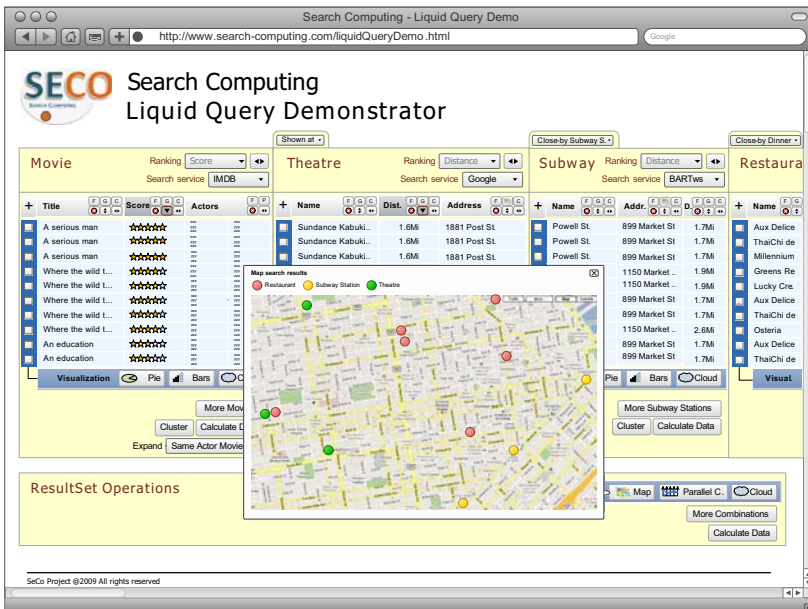


Fig. 13. Mockup of the Map visualization option

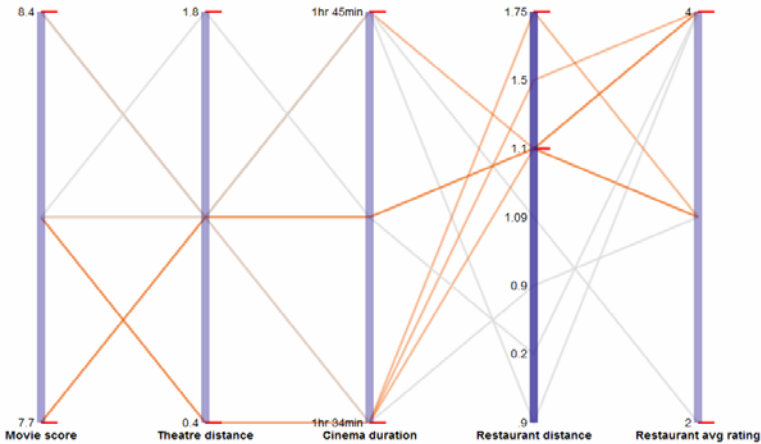


Fig. 14. Mockup of the Parallel Coordinates visualization option



Fig. 15. Mockup of the query management panel

## 5 Conclusions and Future Work

In this chapter we described liquid queries, a user interaction paradigm that exploits the power of SeCo for providing the user with a multi-domain exploratory search environment. During the writing of this chapter, a prototype of liquid queries has been developed for accessing *Yahoo Services* supported by the *Yahoo Query Language* interface; the prototype can be accessed from the SeCo project website and shows a preliminary implementation of several features described in this chapter. A second prototype will soon become available, directly connected to the execution engine, and will demonstrate the running example of Section 4.

Future work includes several directions:

- The implementation of a fully functional prototype, and the integration of advanced data visualization components (e.g., Elastic Lists [13]), to experiment with non-tabular result presentation metaphors. The parallel coordinate system shown in Fig. 14 has already been implemented and is a first result in this direction.

- The investigation of different heuristics for improving the quality of the result set; the interface will be instrumented with metrics fields showing the different quality measures associated with the current result set (e.g., relevance loss, result set diversity, diversity between combinations, etc.). The user will be able to play with the different forms of trade-offs and the interface will immediately reflect the impact of a choice on the quality of the result set.
- The analysis of the user's interaction with the interface to automatically infer preferences that could be applied to the personalization and optimization of both the query and the result set: e.g., automatically expanding a query (e.g., adding a specific category to the hotel selection criterion if the past interaction reveals a preference for a specific class of accommodation); automatically selecting a service interface among alternative ones based on past user's choices; and automatically configuring the result set presentation (e.g., by automatically charting geo-referenced values if the user normally does so).
- The testing of the user interface, to assess its effectiveness in supporting information seeking and exploratory tasks. The testing will necessarily use a mix of techniques used for top-k query and exploratory systems; the former case require building a set of benchmark queries for which the most relevant results are known a priori, e.g., from expert's evaluation; the latter necessarily rely on user's studies, conducted both in laboratory and on the real scale [31].

## References

- [1] Aula, A., Russell, D.M.: Complex and Exploratory Web Search. In: Information Seeking Support Systems Workshop (ISSS 2008), Chapel Hill, NC, USA, June 26-27 (2008)
- [2] Baeza-Yates, R.: Applications of Web Query Mining. In: Losada, D.E., Fernández-Luna, J.M. (eds.) ECIR 2005. LNCS, vol. 3408, pp. 7–22. Springer, Heidelberg (2005)
- [3] Barbosa, L., Freire, J.: Siphoning hidden-web data through keyword-based interfaces. In: SBBD 2004 (XIX Simpósio Brasileiro de Bancos de Dados, 18-20 de Outubro, Brasília, Distrito Federal, Brasil, pp. 309–321 (2004)
- [4] Bozzon, A., Brambilla, M., Fraternali, F.: Conceptual Modelling of Multimedia Search Applications Using Rich Process Models. In: ICWE 2009, pp. 315–329 (2009)
- [5] Brambilla, M., Cabot, J., Grossniklaus, M.: Modelling safe interface interactions in web applications. In: Laender, A.H.F. (ed.) ER 2009. LNCS, ch. 29, vol. 5829, pp. 387–400. Springer, Heidelberg (2009)
- [6] Broder, A.: A taxonomy of web search. SIGIR Forum 36(2), 3–10 (2002)
- [7] Cafarella, M.J., Halevy, A., Zhang, Y., Wang, D.Z., Wu, E.: WebTables: Exploring the Power of Tables on the Web. In: Proceedings of the VLDB Endowment, August 2008, vol. 1(1), pp. 538–549 (2008)
- [8] Clusty (2009), <http://www.clusty.com>
- [9] Dash, D., Rao, J., Megiddo, N., Ailamaki, A., Lohman, G.: Dynamic faceted search for discovery-driven analysis. In: Proceeding of the 17th ACM Conference on information and Knowledge Management, CIKM 2008, Napa Valley, California, USA, October 26-30, pp. 3–12. ACM, New York (2008)
- [10] DBPL Faceted Search (2009), <http://dblp.l3s.de>
- [11] Google Fusion Tables (2009), <http://tables.googlelabs.com/>
- [12] Google Squared (2009), <http://www.google.com/squared>



- [13] Elastic Lists (2009),  
[http://well-formed-data.net/experiments/elastic\\_lists/](http://well-formed-data.net/experiments/elastic_lists/)
- [14] Freebase Parallax (2009), <http://www.freebase.com/labs/parallax/>
- [15] HAKIA (2009), <http://hakia.com/>
- [16] Hunch (2009), <http://www.hunch.com/>
- [17] Inselberg, A.: The Plane with Parallel Coordinates. *Visual Computer* 1(4), 69–91 (1985)
- [18] Jansen, B.J., Booth, D.L., Spink, A.: Determining the user intent of web search engine queries. In: *WWW 2007*, pp. 1149–1150 (2007)
- [19] Jansen, B.J., Pooch, U.W.: A review of Web searching studies and a framework for future research. *JASIST* 52(3), 235–246 (2001)
- [20] Kules, B., Capra, R., Banta, M., Sierra, S.: What do exploratory searchers look at in a faceted search interface? In: *JCDL 2009*, pp. 313–322 (2009)
- [21] Kumar, R., Tomkins, A.: A Characterization of Online Search Behaviour. *Data Engineering Bulletin* 32(2) (June 2009)
- [22] Lee, U., Liu, Z., Cho, J.: Automatic identification of user goals in Web search. In: *WWW 2005*, pp. 391–400 (2005)
- [23] Marchionini, G.: Exploratory search: from finding to understanding. *Commun. ACM* 49(4), 41–46 (2006)
- [24] Microsoft Bing (2009), <http://www.bing.com/>
- [25] Minack, E., Demartini, G., Nejd, W.: Current Approaches to Search Result Diversification, L3S Technical Report,  
<http://www.l3s.de/web/upload/documents/1/paper-camera.pdf>
- [26] Pirolli, P., Stuart, K.C.: Information Foraging. *Psychological Review* 106(4), 643–675 (1999)
- [27] Rajaraman, A.: Kosmix: High Performance Topic Exploration using the Deep Web. In: *Proceedings of the VLDB Endowment*, August 2008, vol. 2(1), pp. 1524–1529 (2009)
- [28] Rose, D.E., Levinson, D.: Understanding user goals in Web search. In: *WWW 2004\_ Proceedings of the 13th international conference on World Wide Web*, New York, NY, USA, pp. 13–19 (2004)
- [29] Sacco, S.M., Tzitzikas, Y.: *Dynamic Taxonomies and Faceted Search, Theory, Practice, and Experience*. The Information Retrieval Series, vol. 25, p. 340. Springer, Heidelberg (2009)
- [30] Shafer, J.C., Agrawal, R., Lauw, H.W.: Symphony: Enabling Search-Driven Applications. In: *USETIM (Using Search Engine Technology for Information Management) Workshop, VLDB Lyon (2009)*
- [31] White, R.W., Muresan, G., Gary, M.: *ACM SIGIR Workshop on Evaluating Exploratory Search Systems*, Seattle (2006)
- [32] White, R.W., Drucker, S.M.: Investigating behavioural variability in web search. In: *16th WWW Conf.*, Banff, Canada, pp. 21–30 (2007)
- [33] White, R.W., Roth, R.A.: Exploratory Search. Beyond the Query–Response Paradigm. In: Marchionini, G. (ed.) *Synthesis Lectures on Information Concepts, Retrieval, and Services Series*, vol. 3. Morgan & Claypool, San Francisco (2009)
- [34] Wolfram Alpha (2009), <http://www.wolframalpha.com/>
- [35] Yahoo! SearchMonkey (2009),  
<http://developer.yahoo.com/searchmonkey/>
- [36] Yahoo! Pipes (2009), <http://pipes.yahoo.com/pipes/>
- [37] YQL: Yahoo! Query Language (2009), <http://developer.yahoo.com/yql/>

# Chapter 14:

## Building Search Computing Applications

Alessandro Bozzon, Marco Brambilla, Stefano Ceri, Francesco Corcoglioniti,  
and Nicola Gatti

Politecnico di Milano, Dipartimento di Elettronica ed Informazione,  
V. Ponzio 34/5, 20133 Milano, Italy  
{alessandro.bozzon, marco.brambilla, stefano.ceri,  
nicola.gatti}@polimi.it,  
francesco.corcoglioniti@gmail.com

**Abstract.** Search Computing aims at opening the Web to a new class of search applications, by offering enhanced expressive and computational power. The success of Search Computing, as of any technical advance, will be measured by its impact upon the search industry and market, and this in turn will be highly influenced by reactions of Web users and developers. It is too early to anticipate such reactions – as the technology is still “under construction” – but this chapter attempts a first identification of the possible future players in the development of Search Computing applications, by grossly identifying the roles of “data source publishers” and of “application developers”, and by discussing how classical advertising-based models may support the new applications. This chapter also describes the high-level design of the prototyping environment that is currently under development and how the design will support the deployment upon high performance architectures. Finally, we describe advertising as the prevalent business model of the search engines industry, and briefly discuss the options for the evolution of such model in the context of Search Computing.

**Keywords:** Search Computing, software engineering, development process, advertising models, cloud computing, software architectures.

## 1 Introduction

The distinguishing feature of Search Computing is the ability of combining, at query execution time, knowledge extracted from various domain-expert Web sources, thus yielding to knowledge that is more accurate and complete than the knowledge available to general-purpose search systems. Such expertise (about cultural events, medical specializations, popular rock songs, and so on) is contributed through either social processes (e.g., rating, tagging, commenting) or a long and careful knowledge construction process by experts. At the current state of the art, multi-domain queries over such engines can be answered only by patient and expert users, whose strategy is to interact with specialized engines one at a time, and feed the result of one search in input to another one, reconstructing answers in their mind.

With the advent of service computing and the growing interest for the Web as the predominant interface for any human activity, we expect such knowledge to become more and more exposed in the form of search services. But the mere composition of such services by sequential invocation will not solve multi-domain queries, as their interplay usually requires a lot of expertise, especially in handing and composing the search results. This challenged us in thinking to a new technology, built upon five pillars (ad hoc service definition, query optimization framework, ranking methods for join results, execution engine, and liquid queries) that collectively resolve the technical issues of Search Computing.

In this chapter, we analyze Search Computing from a broader, usage- and business-oriented perspective by addressing Search Computing applications. A *Search Computing application* is a vertical Web search application that leverages on the SeCo framework for enabling multi-domain search capabilities. The application concretely resides on a SeCo installation and consists of a configuration of one or more multi-domain queries over the existing service repository.

The chapter is made up with four main contributions. First, Section 2 presents the roles involved in the development of SeCo applications and the development process; subsequently, Section 3 describes the SeCo development environment, comprising a set of tools that support the users in their activities. Section 4 describes the SeCo reference architecture, which has been designed with the objective of being extensible, portable, and deployable upon high-performance architectures. Finally, Section 5 discusses plausible business models that could facilitate the spreading and sustainability of SeCo applications: these include advertising models and the possibility of attracting users or developing new user communities.

These contributions provide essential ingredients for building SeCo applications, but are mutually independent; therefore they are considered in four distinct sections. Sections 3-5 also include a state-of-the-art in the respective fields.

## 2 Development of Search Computing Applications

In this section we identify the main roles involved in the development of Search Computing applications and we describe the development process.

### 2.1 User Roles

Search Computing applications involve users with several roles and expertise for their configuration and usage. In this section, we identify the set of user roles involved in the development of SeCo applications, and we clarify their responsibilities and the required skills. Some roles fall outside the strict SeCo application development process, in the sense that they work for preparing the SeCo environment, in terms of platform deployment and search service development. These roles are:

- **SeCo Experts:** they are software architects that are able to deploy and configure SeCo engines over high-performance computing systems and support SeCo publishers and application developers.

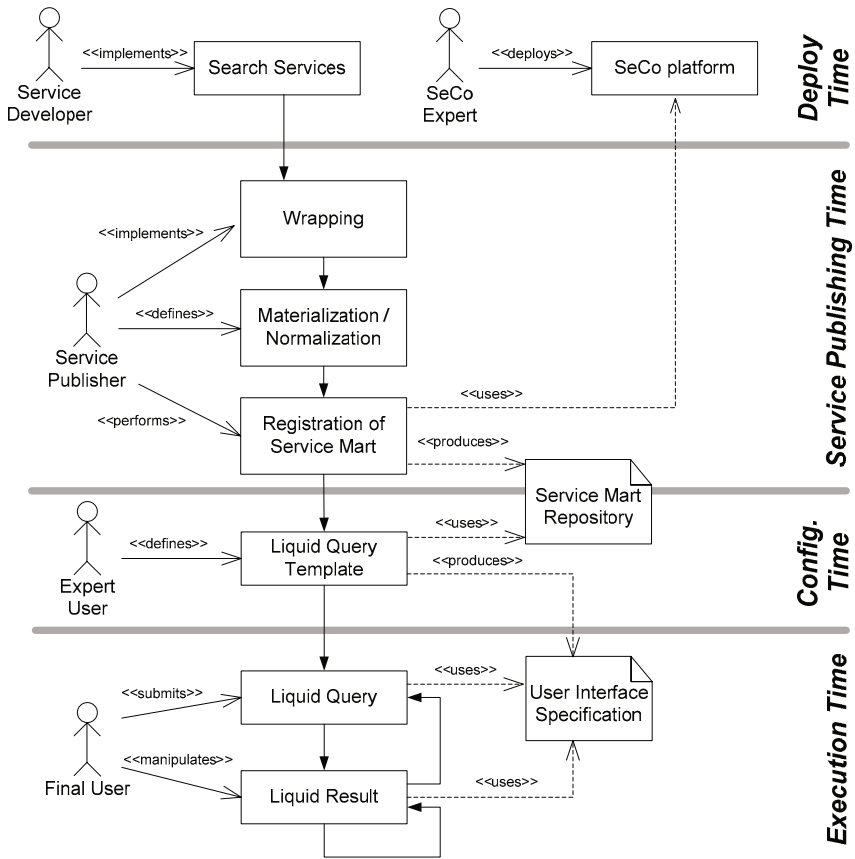


Fig. 1. Development process for SeCo applications

- Service Developers:** they are third party software producers that publish search services on the Web. They independently produce artifacts that are needed for the SeCo applications to run correctly, but are not aware of Search services consist of any kind of REST services, SOAP services, or Web applications that can produce a ranked list of results.

Some other roles are directly involved in the SeCo application development process:

- Service Publishers:** they are in charge of implementing mediators, wrappers, or data materialization components so as to make service interfaces compatible with the SeCo framework, and then register them within the SeCo service repository, thus defining their abstract representations in terms of service marts, access patterns, and connection patterns. Mediators adapt services that are published on the Web. Wrapping technologies span from complex wrapping tools that expose deep Web contents, to simple XSLT transformations for XML documents, to Java classes that introduce ranking and/or chunking features in services. Data

materialization tools are used to transform third party data so as to enable their publication, e.g. files, Excel sheets, or databases; these can be locally stored or acquired within the SeCo architecture, because the features of the service are too poor for granting proper treatment of the data;

- **Expert Users (or Application Developers):** they configure Search Computing application, by selecting the service marts of interest, the respective connection patterns, and associated user interfaces for query and result visualization. They also choose the complexity of the interaction interface, in terms of controls and configurability choices to be left to the user. At runtime, they interact with applications at a high level of sophistication, by composing queries on-the-fly and by executing them.
- **Final Users:** they use SeCo engines to navigate query/result interfaces devised by expert users. They interact by submitting queries, reading results, and refining/evolving them according to the liquid query philosophy.

The peculiar features of SeCo applications require new roles with respect to traditional application development. The most prominent ones are service publishers and expert users. Due to their novelty, no widespread user communities currently exist for these roles; however, they are crucial for the success of SeCo, and therefore some actions must be taken to foster the flourishing of such communities, providing them with suitable methods and tools, as we will discuss in the remainder of the chapter.

## 2.2 Development Process

The development process (shown in Fig. 1) is split into four main development steps:

- **Deployment Time:** this phase consists in the actual development of search services and the deployment of the SeCo platform on the suitable infrastructure. The service development and deployment is delegated to external developers and is conceptually independent from any subsequent step within the SeCo framework. The deployment of the SeCo platform as well is assigned to SeCo experts and is performed once and for all, independently on the number of actual SeCo applications that will run upon it;
- **Service Publishing Time:** several activities are needed for publishing search services within the SeCo framework: definition of the service wrappers, possible specification of materialization design for the retrieved data, normalization of the data, and registration as service marts in the SeCo service repository;
- **Application Configuration Time:** this phase, in charge to the Expert User, consists in selecting the Service marts of interests and the corresponding details, such as the connection patterns, the parameters of interest, and so on. Subsequently, the expert user defines a liquid query template for a specific SeCo application, which entails the specification of the user interface aspects. In particular, the expert user defines the structure of the liquid queries in terms of query templates that will be completed at runtime by the end user. A liquid query template is composed of:

1. a set of service interfaces;
  2. a set of connection patterns for joining the involved service interfaces;
  3. a set of additional selection or join predicates;
  4. a default ranking function defined over the scores of service interfaces;
  5. a set of possible sorting, grouping, and clustering attributes that can be applied on the extracted result set;
  6. a set of positive integer values  $K$  that represent the possible sizes for the result pages;
  7. a set of available query expansions, defined next.
- **Application Execution Time:** in this last phase, the Final User can navigate the application, i.e., the queries and results, and possibly applies some configuration details. At runtime, the end user is presented with a Liquid query Template that he can fill in with the actual query parameter. In addition, several parameters of the query template, which are initially set to defaults, can be tuned; these include:
    1. the projection attributes that define the information visible to the user;
    2. the ordering of services and of their attributes within the query;
    3. the choice of the weights of the scores of service interfaces involved in the query;
    4. the choice of cluster attributes to be used to initially visualize the query results;
    5. the optional grouping attribute to be used to initially group the query results;
    6. the choice about the size and production (e.g., continuous or chunked) of results in the result page.

Since Liquid Query vision is towards continuous evolution, manipulation, and extension of queries and results, according to the “search as a process” paradigm, the query lifecycle consists of iterations of the steps of **query submission**, when the end user submits an initial liquid query; **query execution**, producing a liquid result that is provided to the user interface; and **result browsing**, when the result can be read and manipulated through appropriate interaction primitives, which update either the liquid result or the liquid query. Depending on the kind of user interaction, the query execution performed by the engine might be suspended, restarted, or stopped. If the interaction only involves reshaping of available data, the engine may not be involved in the needed actions and the information is manipulated at user interface level.

The development process takes into account the trend towards empowerment of the user, as witnessed in the field of Web mashups (see Chapter 5 and [10]). Indeed, only the basic tasks that deal with service development (performed by service developers) require actual programming expertise. All the other design activities are moved to service registration time and to application configuration time, so that designers only need a conceptual understanding of services and queries, and do not need to perform low-level programming.

### 3 Development Tools

Several peculiar aspects affect the development process and the needed tools for SeCo applications:

- The **need for components provided by third parties** (in particular: search services): this implies that the process includes the need of scouting and investigating about the ecosystem of existing services within the domains of interest, for publishing and registering the found services.
- The **vertical focus** of SeCo applications: starting from the repository of available Search Services, canned interfaces can be devised for implementing verticals requiring specific domains, whose services are made available in a rich number.
- The need for **configurability of the applications**: the continuous evolution of several pieces of the architecture (services, tags and descriptions, interfaces, results) makes several steps of the development more conveniently located at query deployment time instead of service registration time.

As highlighted in the development process in Section 2.2, these features push **towards empowerment of the user** and ask for specific tools for supporting the developers. In this section we discuss the features of the existing web development tools, we highlight which of them can be borrowed for SeCo and we describe our vision towards instrumentation of the SeCo development process.

### 3.1 Web Design Tools and Environments

In the context of web application design, developers and designers usually exploit commercial or open source tools for performing their job. In this section we identify the classes of tools that are currently in use for Web application design, considering three main dimensions:

- **Target Users:** analysts, developers, and visual designers;
- **Development Focus:** database-oriented, service-oriented, user interface-oriented, and search-oriented;
- **Tool Availability:** local or remote.

#### 3.1.1 Target Users

With respect to target users, we identify three main approaches, which correspond to the respective user roles:

- **Analysts and Designers:** this user role typically works with **Model-driven design tools**. Such tools include BPM (business process modeling) tools, Web engineering tools based on conceptual models, UML design tools, and MDD/MDA based techniques. Notable BPM tools that provide good Web deployment features include Oracle BPM<sup>1</sup>, WebRatio BPM<sup>2</sup>, and BillFish BPM<sup>3</sup>. The most known representative of Web engineering tools that exploit conceptual modeling and formalized development process is WebRatio<sup>4</sup>, while a good choice of UML modeling and partial code generation (also for the web) exists. Among

---

<sup>1</sup> <http://www.oracle.com/us/technologies/bpm/index.htm>

<sup>2</sup> <http://www.webratio.com/>

<sup>3</sup> <http://www.billfishsoftware.com/>

<sup>4</sup> <http://www.webratio.com/>

them, we can mention MagicDraw<sup>5</sup>, IBM Rational<sup>6</sup>, and others. These tools provide a visual design environment that allows drawing conceptual models of the application, to debug and apply some validation, and to generate running code. Coverage of the various aspects of the application design and completeness of code generation depend on the tool. Typically, UML tools generate stub classes corresponding to the design and then provide hooks to IDEs for completing the implementation. Some BPM tools provide automatic generation of the running prototype of the web application, while more sophisticated model-driven tools like WebRatio provide refined modeling primitives that allow going for full code generation of the final application. Fig. 1 shows the WebRatio interface for designing the Web application hypertext and the contextual menu that allows the user to immediately see the generated Web application page corresponding to the selected modeling concept. The features that can be borrowed for SeCo tools include: *visual composition* of the applications (e.g., at service registration time for mapping to existing service marts; at application configuration time for selecting the marts of interest and composing them) and *automatic deployment* of the running prototype.

- **Software Developers and Debuggers:** this roles work with **Code-driven development**. This paradigm collects IDEs (Integrated Development Environment) that are explicitly targeted to web development or that covers general-purpose development but include some features or plugins addressing web issues. This class includes a set of diverse products, spanning from Eclipse WTP project<sup>7</sup>, which provides a set of Eclipse plugins for Web applications development, to Microsoft Visual Studio. The main features that can be borrowed for SeCo tools are: *code-level support* for building and debugging service wrappers and the *code-level refinement* of the application through code inspection (e.g., for configuration files).
- **Visual Designers:** this role works with an **interface-driven approach**, developing Web interfaces with attention to detailed graphical appearance. Examples of tools that support this development approach include authoring tools, solutions like Adobe DreamWeaver<sup>8</sup>, Aptana Studio<sup>9</sup>, and a plethora of commercial and freeware HTML editors. As an example of interface, Fig. 3 shows the command panels of Dreamweaver. Further examples are described in the next section. The main feature that can be borrowed for SeCo tools is the support for *graphics and interface customization*, e.g., for complying with customers' visual identities.

A separate category is represented by the mashup development tools, which are of high relevance for SeCo. This category is not analyzed here because it has been widely addressed in Chapter 5. The main feature that can be borrowed for SeCo is the *online availability* of the design tools.

---

<sup>5</sup> <http://www.magicdraw.com/>

<sup>6</sup> <http://www-01.ibm.com/software/awdtools/developer/rose/index.html>

<sup>7</sup> <http://www.eclipse.org/webtools/>

<sup>8</sup> <http://www.adobe.com/products/dreamweaver/>

<sup>9</sup> <http://www.aptana.org/>



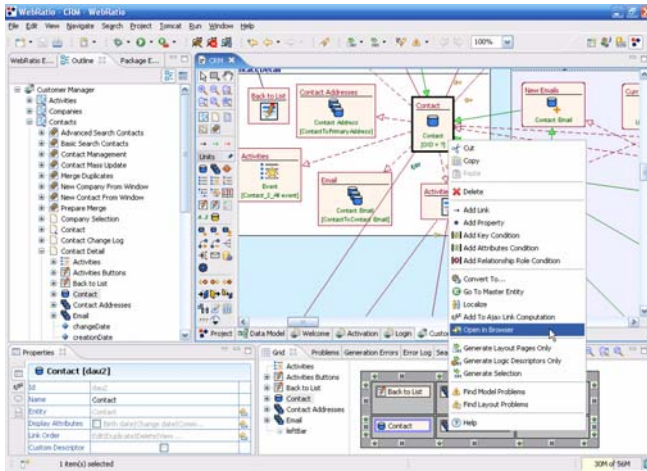


Fig. 2. WebRatio modeling interface and link to the generated Web page

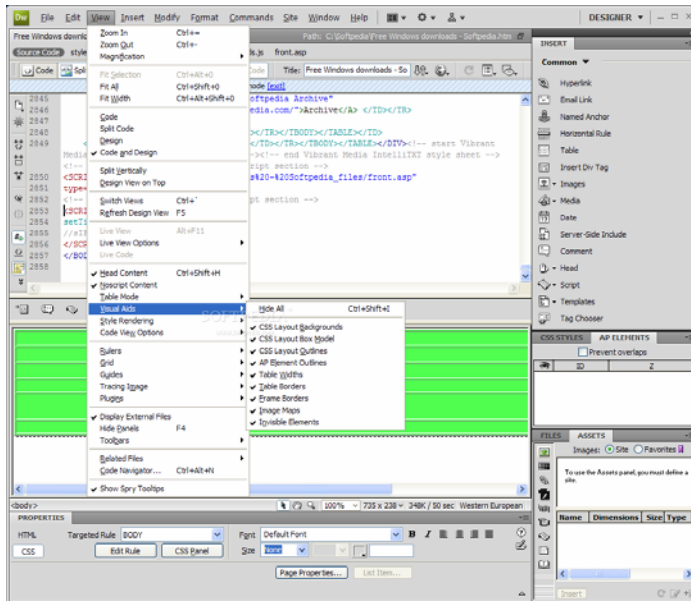


Fig. 3. Adobe Dreamweaver CS4, HTML design interface

### 3.1.2 Development Focus

Most of the development tools for the Web start from a specific perspective to the problem. **Interface- and interaction-oriented tools** root into the hypermedia field; they include tools like Adobe CS4<sup>10</sup> and Adobe Flex<sup>11</sup>, which deliver high quality animations, interfaces, and rich applications.

<sup>10</sup> <http://www.adobe.com/products/creativesuite/>

<sup>11</sup> <http://www.adobe.com/products/flex/>

**Database-oriented tools** start from the opposite point of view, by allowing the design of Web applications upon published data sources. Such tools include Caspio Bridge<sup>12</sup>, WyaWorks<sup>13</sup>, Zoho Creator<sup>14</sup>, Dabble DB<sup>15</sup>, Trackvia<sup>16</sup>, and several other similar solutions. They all allow to build Web applications made of forms, lists, and data details starting from a database schema or other data sources (e.g., spreadsheets, text files, and so on). They typically provide application templates for popular needs too (e.g., CRM, accounting, project management, and so on). Current trends move towards full-fledged online database platforms that allow publishing and management of online data sources. Most of them provide online development interfaces and Software as a Service business models. The main similarity to SeCo is the schema-based definition of services and results, as well as the structured specification of search queries.

Finally, **service-oriented tools** consider services (instead of data sources) as first class citizens for the web application. This class comprises Web service orchestration tools, mashup tools, and service repositories and registration tools. The former can be classified into two main subcategories: service orchestration tools, like Oracle SOA<sup>17</sup> suite (comprising a BPEL process manager, a service bus, business rules and code editors), Oracle WebLogic<sup>18</sup> suite (an application server specifically targeted to Web services, formerly owned by BEA), ActiveVos<sup>19</sup>, JOpera<sup>20</sup>, and others, whose aim is to specify executable orchestrations of services based on BPEL; and more general tools, that can be referred to as BPM tools, including Oracle BPM (born from the Collaxa BPEL engine, acquired in 2004), IBM BPM<sup>21</sup>, BizAgi<sup>22</sup>, and others. These tools allow the designer to describe service interactions at a more abstract level through workflow models (for instance, based on the BPMN notation). In some sense, various SeCo features refer to this vision: the *service-based invocation* paradigm, the *collaboration* between services for achieving a common result, and the *orchestration* of the query plans for producing search results.

We don't dig into the categories of mashup and service registration tools, since they have been widely discussed in Chapter 5 and Chapter 9 respectively. An example of tool at the verge between mashups and BP specifications is JOpera, that supports quick composition and orchestration of services, as well as monitoring of execution. Fig. 4 shows a sample screenshot of the tool.

---

<sup>12</sup> <http://www.caspio.com/bridge/>

<sup>13</sup> <http://www.wyaworks.com/>

<sup>14</sup> <http://creator.zoho.com/>

<sup>15</sup> <http://www.dabbledb.com/>

<sup>16</sup> <http://www.trackvia.com/>

<sup>17</sup> <http://www.oracle.com/technologies/soa/soa-suite.html>

<sup>18</sup> <http://www.oracle.com/appserver/index.html>

<sup>19</sup> <https://www.activevos.com/>

<sup>20</sup> <http://www.jopera.org/>

<sup>21</sup> <http://www-01.ibm.com/software/info/bpm/>

<sup>22</sup> <http://www.bizagi.com/>

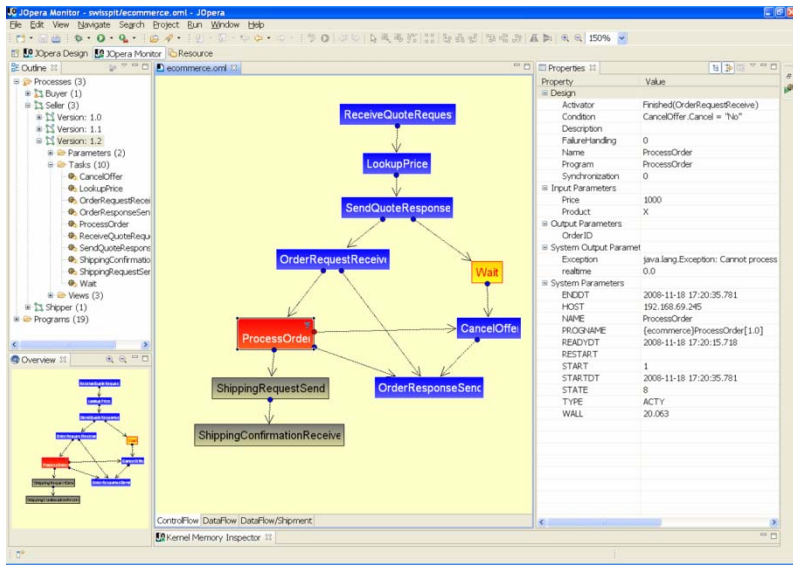


Fig. 4. JOpera Web service composition screenshot

Another emerging category of tools is related to **search-based application development**. With the increase of sophistication and the diversification of requirements that modern search solutions exhibit, the need arises of unbundling the functionality of a search system into a set of reusable components, which could be integrated to produce a variety of solutions based on the paradigm of search. One example is the Symphony platform by Microsoft, which enables non-developers to build and deploy search-driven applications that combine their data and domain expertise with content from search engines and other Web Services [23]. The similarity to SeCo is quite straightforward, although some basic features (such as join of results) are still missing in existing tools.

Other approaches to search-based development target the skilled software developer. Google Base API<sup>23</sup> relies on APIs for allowing developers to design their search applications. It allows to combine unstructured (i.e., full-text based) and structured (i.e., exploiting a data schema) queries and to update contents in the form of Google Data API feeds. It supports multiple ranking, overcrowding removal (thus avoiding to provide several similar items in the same result set), adjusted text results, suggestions on result schema, and much more. For example, the following query combines full-text search on digital cameras and structured search on brand “Canon”:

```
snippets?q=digital+camera&bq=[brand:canon]
```

Google Base API are exposed as REST services invoked through HTTP GET, like in the following example:

```
http://www.google.com/base/feeds/snippets?bq=[brand:canon]
```

<sup>23</sup> <http://code.google.com/apis/base/>

Yahoo Query Language (YQL)<sup>24</sup>, instead, allows to query, filter, and combine data from different sources across the Internet through SQL-like statements. The following YQL statement, for example, retrieves a list of cat photos from Flickr:

```
SELECT * FROM flickr.photos.search WHERE text="cat".
```

YQL is also available as a REST Service that can be invoked through HTTP GET, passing the YQL statement as a URL parameter. For instance:

```
http://query.yahooapis.com/v1/public/yql?q=SELECT * FROM flickr.photos.search WHERE text="Cat"
```

When it processes a query, the YQL service accesses a datasource on the Internet according to a given access specification, transforms its data, and returns the results in either XML or JSON format. YQL can access several types of datasources, including Web services, REST API and Web content in formats such as HTML, XML, and RSS.

These APIs are extremely useful for SeCo, since they can be wrapped and exploited as *providers of search services*.

### 3.1.3 Tool Availability

A crucial aspect in modern Web application development is how development tools are made available to developers. Two major categories can be identified: tools that are **available online** with SaaS (Software as a Service) model, and tools to be **installed locally** on the developer's machine.

Among the tools available online we can mention: most mashup tools (see Chapter 5), some recent database-driven (like WyaWorks) and interface-driven design tools, the large class of CMS (Content Management System) tools, like Drupal<sup>25</sup> and Joomla<sup>26</sup>, and hybrid solutions like App2You<sup>27</sup>, which stands in between database-driven and interface-driven tools.

Desktop development tools include heavy weight solutions like Eclipse, Adobe CS4 suite, Microsoft Visual Studio<sup>28</sup>, Webratio, and so on.

## 3.2 SeCo Development Tools

To comply with the SeCo vision, we foresee a set of tools to be provided to developers for covering the lifecycle phases. For SeCo application development, tools are crucial for *service registration*, *application configuration*, and *query plan tuning*, while tools for *service development* are outside the scope of the framework and interfaces for *application execution* are described in the Liquid Query approach (Chapter 13).

---

<sup>24</sup> <http://developer.yahoo.com/yql/>

<sup>25</sup> <http://drupal.org/>

<sup>26</sup> <http://www.joomla.org/>

<sup>27</sup> <http://app2you.com/>

<sup>28</sup> <http://www.microsoft.com/visualstudio/>

**Service registration tools** will consist of a set of facilities for allowing normalization of service interfaces and their registration as Service Marts. Tools supporting the normalization will help in:

- *Defining the Service Mart signature;*
- *Defining the Access Pattern structures;*
- *Defining the normalized schema* of the underlying data model, structured in terms of primary table and SeCondary tables as described in Chapter 9;
- *Specifying the service interfaces*, in terms of ranking, chunk, cache, and cost descriptors;
- *Defining the annotations* of the services, in terms of reference domain and keywords;
- *Establishing connection patterns* between pairs of service marts and service interfaces, to describe possible join paths for queries.

The tools will feature mapping-based interfaces that will allow picking elements from the service input/outputs (and domain descriptions) and populating the conceptual models.

**Application configuration tools** will allow composing application structures consisting of sets of connected service marts. The tools will support the following activities:

- *Exploring* the service repository, through visual navigation;
- *Selecting* the services of interest for the application and the respective connection patterns, including the ones needed for query expansions;
- *Defining* the interface of the query submission form and of the resultset, together with the default settings for the application and the allowed Liquid Query operations.

**Query plan tuning tools** will consist of a visual modeling environment that allows developers to edit query plans specified according to the Panta Rhei notation (as described in Chapter 12). Such plans are usually automatically generated by the plan optimizer, but advanced developers may want to manually refine them to take in consideration domain specific knowledge or customized choices that are not available to the optimizer.

All the tools will be developed as online applications, at the purpose of increasing SeCo application design productivity, reducing the time to deployment, and avoiding the burden of downloading and installing software.

## 4 Software Architecture

This section describes the architectural issues involved in the development of SeCo systems. Being SeCo a Web system dealing with a large amount of concurrent end user requests, sub-second *response time* and *scalability* are of primary importance. Therefore, high-performance architectures and deployment environments able to satisfy these requirements must be part of the solutions.

#### 4.1 High-Performance Architectures for Web Applications

Web applications usually adopt a three-tier architecture, comprising *presentation*, *business logic*, and *data*. The data tier is usually based on a database, while in SeCo applications it consists of the registered remote services being invoked by the query engine. Scalability in Web applications can be achieved by using more powerful server machines (*vertical scalability*) or by allocating multiple server machines organized in a cluster (*horizontal scalability*)[5]. *Cluster computing* [22] enables the management of an increased traffic by splitting incoming requests to multiple servers, exploiting the fact that most user requests can be handled independently, as it happens with Search Computing queries. Different *load balancing* techniques have been devised [8] to achieve an even utilization of computation nodes. By allocating redundant nodes to replace failing machines, *failover clusters* can be used to provide *high availability* of deployed applications.

A promising deployment environment for Web applications is provided by *Cloud computing* [2] [7]. According to this paradigm, Web applications are deployed to a set of virtualized, interconnected storage and computing resources offered by third-party providers, globally referred to as a cloud to abstract from their physical location and characteristics. A cloud deployment environment (such as Amazon EC2 [1]) offers several benefits to the application provider, among which the possibilities to (1) dynamically allocate resources to an application, thus being able to dynamically scale it up to increased workloads, and (2) to eliminate fixed costs related to in-house provision of the application, paying only based on the usage of offered resources.

Short *response time* and *time-to-screen* are crucial to guarantee system responsiveness. These parameters are affected by two main factors in SeCo: internal *query processing time* and remote *services invocation time*. The former can be reduced by executing a query on multiple nodes in parallel, by exploiting *inter-query* and also *intra-query* parallelism. SeCo queries running on multiple nodes can be assimilated to *distributed queries* in a database setting [15], where a single query plan is divided into a set of sub-plans, scheduled and executed on different database nodes. However, intra-query scheduling in Search Computing is simpler (because there is no need of considering allocation of data) and can benefit from existing *scheduling algorithms* (e.g., the *work stealing* algorithm [4]) developed in the field of *grid computing* [6]. Another popular paradigm for parallel processing is *Map-Reduce* [11], a framework for efficiently distributing and scheduling computations expressed using map and reduce primitives. While Map-Reduce has proven useful for batch data processing (e.g., building a search engine index), its programming paradigm makes the execution of relational joins cumbersome; nonetheless, an extension – *Map-Reduce-Merge* [24] – has been proposed to address this issue.

*Service invocation time*, instead, can be reduced by minimizing and optimizing communications with services, possibly avoiding them at all. At a physical level, invocation times can be reduced by efficiently using available communication protocols. HTTP, in particular, provides facilities for *caching* Web server responses and *pipelining requests* to Web servers [12]. At an higher level, the communication problem has been addressed in *metasearch systems*, where a *crawl-metasearch hybrid approach* [9] has been proposed to reduce Web search costs, by indexing low-turnover and small data sources while meta-searching the other ones. A similar approach can be adopted for Search Computing, by recurring to *materialization* (see

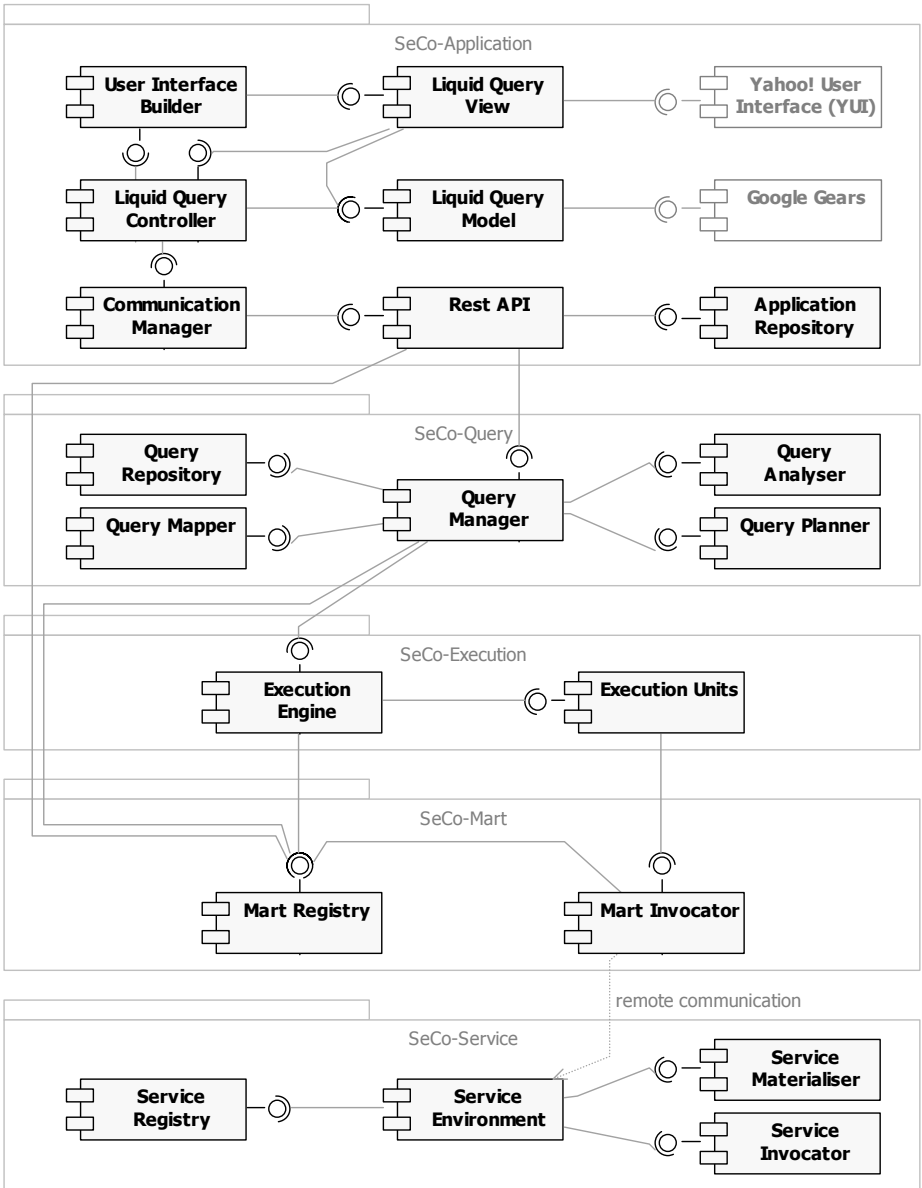


Fig. 5. UML component diagram showing the logical system architecture

Chapter 9) of frequently accessed services that provides access to small amounts of data changing infrequently. A different approach is represented by *distributed workflow systems* [20], where service nodes directly participate to the orchestration process in a peer-to-peer fashion, thus eliminating the central orchestrator bottleneck. The latter approach however is not applicable to SeCo, since it would require service providers to actively support the Search Computing framework.

## 4.2 SeCo Architecture

This section describes the reference software architecture designed to support the runtime execution of Search Computing queries. Fig. 5 shows the logical architecture of the system, expressed in terms of software component to be deployed (and possibly replicated) on different execution nodes. As shown in the figure, the architecture is divided in five layers:

- The lower layer, called **SeCo-Service**, is used by service developers and offers facilities to wrap and expose existing services. A *Service Registry* hold wrapper and concrete service descriptions, as described in Chapter 9. The *Service Engine* handles runtime invocations by means of a *Service Invocator* component that abstracts the service physical details.
- The **SeCo-Mart** level provides the service mart abstraction consisting of the *Service Mart Registry* and *Invoker* components, which respectively store descriptions of service marts and interfaces, and support the invocation of the latter according to the standard HTTP+JSON interface described in Chapter 9.
- The core level, called **SeCo-Execution**, contains the execution engine made of a core *Engine* component and of a set of *Execution Units* realizing the *Panta Rhei* model. The latter are programmed, installed, and tuned by SeCo experts.
- The **SeCo-Query** layer includes all the components required for processing a query. The *Query Mapper* decomposes natural language queries into domain-aware subqueries. The *Query Analyzer* performs the selection of access patterns service interfaces, thereby producing a service interface-level query<sup>29</sup>. The *Query Planner* translates the query into an optimized *Panta Rhei* execution plan. Queries and optimized plans are stored in a *Query Repository* for subsequent reuse, while the *Query Manager* orchestrates the whole optimization process.
- The **SeCo-Application** level provides a *Rest API* to submit queries, an *Application Repository* to store application-specific data (such as UIs' definitions) and *liquid query* support. Liquid queries are the client-side front-end of the SeCo architecture, designed as a Rich Internet Application [3] so as to enable a fluid user interaction thanks to client-side data management and to asynchronous communications with the SeCo back-end. A standard Web browser incorporates the liquid query application shell, which is an application written in JavaScript and based on a Model-View-Controller design pattern; the application leverages the libraries and functionalities offered by the Yahoo! User Interface (YUI) libraries<sup>30</sup> and by Google Gears<sup>31</sup>. The *Liquid Query Controller* initializes the application, builds the graphical user interface through the *User Interface Builder*, manages the user interactions and the interaction status, and communicates with the SeCo API through the *Communication Manager*. The *Liquid Query Model* is responsible to store and massage client data after each interaction (e.g., applying filtering sorting, aggregation of results, synchronization with a client persistent repository to enable

---

<sup>29</sup> In the current prototype, the query mapper and analyzer are not developed, as we assume that the input query is already described at the level service interfaces.

<sup>30</sup> <http://developer.yahoo.com/yui/>

<sup>31</sup> <http://gears.google.com/>



off-line usage, etc.). Finally, the *Liquid Query View* comprises the graphical objects and presentation properties specific for the SeCo applications. Client-side user interactions are associated to either *local* or *global* operations; the former can be executed directly on the client, the latter require the engine's intervention.

### 4.3 Deployment

This section describes the deployment of software components on processing nodes. As shown in Fig. 6, deployment is organized on three tiers:

- The **Service Tier** consists of the processing nodes providing access to registered services. A *Service Composition and Creation Framework* can be deployed to facilitate the exposing of services, and consists of the component of the *SeCo-Service* layer.

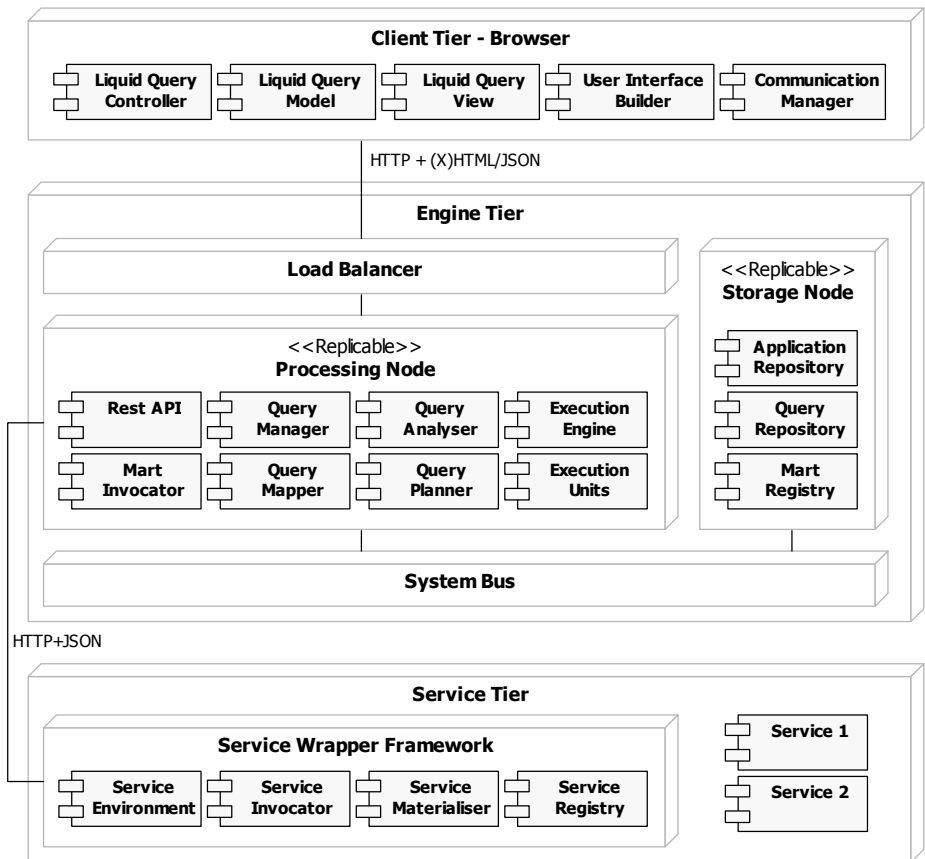


Fig. 6. Deployment of software components

- The **Client Tier** consists of client machines locally running the liquid query UI, which is offered as a JavaScript component running inside Web browsers.
- The **Engine Tier** represents the query engine, which is invoked by clients and executes Search Computing queries over registered services. Engine components can be simply deployed on a single-machine or distributed and replicated across multiple machines to achieve massive scalability. In the latter case, components can be grouped in two types of nodes, namely (1) *processing nodes*, responsible of query execution and (2) *storage nodes*, containing service and query definitions. If deployed on a cloud infrastructure, these two types of nodes can be dynamically replicated with the assistance of a load balancer, in order to cope with increasing workloads. Inter-component communication and coordination are guaranteed by a *System Bus*.

In the prototyping of the Engine Tier, besides testing functionalities, we will soon address crucial aspects such as robustness and scalability. For the second generation of prototypes, we plan to use a space-based middleware, such as GigaSpaces XAP<sup>32</sup>, which represents a promising solution: by decoupling state (stored in space entries) from computation (provided by stateless components) it automatically supports component replication, load balancing and fail-over.

## 5 Business Models in Search Applications

This section discusses plausible business models that could facilitate the spreading and sustainability of SeCo applications. We start with an overview of the advertising strategies in the search field, and then we provide some hints on the possible SeCo advertising models and strategies for attracting users or developing new user communities.

The rapid growth of the Internet is transforming the way information being accessed and used. Newer and innovative models for distributing, sharing, linking, and marketing the information are appearing. As with all communication media, the major source of financial support is advertising [12]. Several Internet advertising formats are commonly used: banners, rich media, email, classifieds, referrals [21]. In today's Internet advertising industry, the so-called *search format* is the most relevant revenue-generating context: advertisers pay search engine companies to list their links (commonly called *sponsored links*) in response to specific search word or phrases. The revenue generated by the search format of advertising constitutes more than 90% of the whole revenues of search engine companies<sup>33</sup>.

In the following three subsections, we describe the economic principles of the search format and subsequently the tools provided by the main search engine companies that can be exploited by advertisers and third parties.

---

<sup>32</sup> <http://www.gigaspaces.com/xap>

<sup>33</sup> About 97% of the income of Google (about 10 billion dollars per year) comes from advertising, the remaining 3% from sales of products [21].

## 5.1 Principles of Advertising in Search Engines

The economic principles of the web advertising search format are simple. The search engine chooses a list of sponsored links, each one composed of a head title, a brief description, and the link, to be shown (impressed) alongside the results of the search and, whenever a user clicks on a sponsored link, the corresponding advertiser [24]. This pricing scheme is commonly called pay-per-click (PPC) and is considered the fairest for search engine and advertisers. It has been shown [21] that the other two schemes, pay-per-impression (PPI) and pay-per-transaction (PPT), advantage the publisher (in this case the search engine) and the advertiser, respectively.

The idea behind the impression of sponsored links alongside search results is that a user could be interested in visiting commercial links that are strictly related to her search; this happens indeed very frequently, and therefore the revenues generated by the search format is very impressive. The choices of the list of the sponsored links to be shown and of the amount of money that a clicked advertiser must pay are accomplished by the search engine in the attempt to maximize its expected revenues, which depend on the probability that a user will click on a link and the amount of money that the corresponding advertiser would pay for that click. Obviously, the larger are such two factors, the larger the expected revenues. This problem is essentially an auction problem and is commonly studied by resorting to microeconomic tools [18].

We focus on how a search engine chooses the list of sponsored links to be shown. Given a search accomplished by a user, the first task that the search engine must address is to determine the most interesting advertisers for the user. This task is accomplished by estimating the click probabilities for each sponsored link. In doing so, the search engine exploits context information (e.g., keywords searched by the user, user's language, country, and IP) and historical data. Essentially, the click probabilities are produced by considering the last (e.g., one thousand) impressions of a sponsored link in the presence of the given context and counting the number of times it has been clicked. These probabilities are commonly called click-through-rates (CTR) and range from 0.5% to 20% with an average around 3% in practical applications.

The basic context information concerns the keywords searched by the user. An advertiser can register for one keyword or for a list of keywords, e.g., “car”, “sport car”, and “luxury sport car”; the more specific is the list of keywords, the easier and the more precise is the targeting of the advertisement to the most interested users. The advertiser can provide additional information, such as the language of the audience, the country, the region, and the city. For example, a bakery in Paris will likely target just the city of Paris, while a nationwide bank in Australia will likely want to target the entire country. The search engine will then determine whom to show a given sponsored link on the basis of several factors, including user's domain, search terms, computer's IP address (estimates its geographical location), and language preference set for the search engine.

The registration of an advertiser for a keyword (or a list) with specific language and location information is concluded by setting the maximum amount of money that the advertiser would pay when the sponsored link is clicked. This value is usually called the advertiser's *bid*. Note that such amount of money is not generally the

amount the advertiser will pay if clicked; rather, it is the largest amount that would be paid. In practical applications, the values are in the range from \$0.05\$ Euros (minimum value acceptable by the search engine) to \$15\$ Euros. In addition to setting such value, the advertiser can choose a maximum budget per day or a maximum number of impressions per day.

On the basis of the context, click probabilities, and advertisers' bids, the search engine chooses the list of sponsored links to be shown. Generally speaking, the search engine maximizes the cumulative revenue expected from each sponsored link. The choice of the amount of money that the clicked sponsored link must pay is an intricate technical issue, and therefore we provide only the general concepts, omitting details. On one hand, the search engine should maximize its revenue by maximizing the payments; on the other hand, it must avoid strategic behaviors of the advertisers that could decrease the search engine's profit. The aim is to produce payment rules that provide the right incentives to the advertisers to bid their true evaluations. In this way, strategic behaviors can be avoided and the economic mechanism behind the auction is said to be *incentive compatible*. The design of the most effective economic mechanism for sponsored search auction is currently an open issue in the microeconomic literature [21].

## 5.2 Advertising Tools in Search Engines

We review the tools provided by the three main search engines: Google, Yahoo!, and Microsoft. For reasons of space, we describe in detail the tools provided by Google and we briefly report the differences between these tools and those provided by Yahoo! and Microsoft.

Google provides several tools for Internet advertising. The basic tool for search format is AdWords [16]. This tool allows an advertiser to register for keywords, specifying language, location information for targeting audience, and upper bounds over budget and impressions. AdWords exploits GoogleMaps for the location information and can add maps to the sponsored links, as the impression of images and maps has been shown to increase the interest of users and consequently the click probability. An advertiser can also select the screen area where the sponsored link will be shown, either on the top of the search results or on the right of them, which are managed by two different auctions.

Auctions are based on the *generalized Second price* (GSP) [21], where the amount of money paid by the sponsored link in position  $i$ -th is the bid of the advertiser whose sponsored link appears in position  $i+1$ -th. In the version implemented by AdWords the price is increased by 0.01 Euro. Although this kind of auction does not produce the right incentives for advertisers to bid their true evaluations (i.e., it is not incentive compatible), it is shown to produce large revenues for the search engine and to avoid price instability in the market. Currently, Google is not interested in employing alternative economic mechanisms that in theory outperform GSP.

Google AdWords provides an advertiser with additional features: advertisers can select the *devices* and the *content networks* where her sponsored links will appear. Relative to the first feature, the advertiser can target either desktop and laptop computers, or iPhones and other mobile devices with full Internet browsers, or both. The Google Content Network [15] allows AdWords to show sponsored links also on

sites that are not search engines, including products like Google Groups and Gmail, as well as other important search sites like AOL and Ask.com, or content sites like NYtimes.com and About.com.

The tool AdSense [15] is used by website owners who wish to make money by displaying sponsored links on their websites. Website owners can use Google AdSense with two different modalities:

- The website owner can publish a Google search frame where a user can search contents through keywords. In addition to the search results, the website owner can then publish on such frame the list of sponsored links, produced by Google AdWords.
- The website owner can publish a frame wherein some sponsored links will be impressed, letting to Google AdSense the choice of best links on the basis of the content reported in the site. More precisely, AdSense analyzes the site by extracting the main keywords and subsequently submits such keywords to AdWords to produce the list of sponsored links.

With both modalities, the revenue received from the advertisements published by website owners is shared with Google. The exact ratio of the money that Google gives to the website owners depends on the specific website and is private information; usually it ranges from 40% to 50%.

Yahoo! and Microsoft provide tools very close to the ones provided by Google. Specifically, Yahoo! Search Marketing [24] is analogous to Google's AdWords and Yahoo! APT [24] is analogous to Google's AdSense. Yahoo! tools exploit the same auction model (GSP), but – differently from Google's tools - they allow advertisers to make their bids in real time. Microsoft Advertising [17] combines the services provided by Google's AdWords and AdSense. The advantages of Microsoft Advertising lay on the media network on which the advertisement can be impressed (in addition to the search engine), which contains high-traffic sites such as Facebook, Digg, Zune, and Windows Live Sharing.

All the main players (Google, Yahoo!, and Microsoft) provide sophisticated strategic tools to advertisers in order to optimize their campaigns. They allow an advertiser to monitor the number of impressions and clicks, to simulate the effects of increments ad reductions of the value of the bids, to monitor information about the users, and so on.

### 5.3 Business Models for Search Computing

In this section, we sketch some ideas about business models for SeCo, by explaining some scenarios for profit sharing among all players within the SeCo environment, and specifically showing the aspects of the advertising though the search format could be extended due to the new aspects of Search Computing.

A Search Computing application relies on the existence of underlying sources. A SeCo developer could decide to act independently from source owners, e.g. by using publicly available sources; then, he should play also the role of SeCo publisher and guarantee the access to the data which are needed for the application. In such a case, the business model of a SeCo application is simple, as there is only one player, the SeCo developer, whose incentive is to build an application as attractive as possible for

its perspective users. SeCo will provide to such player a new application development environment supporting a new class of Web applications, to be compared to the many environments already available.

However, the most interesting scenario for source computing is one where the SeCo developer acts as a “broker of information”, by attracting content owners to participate to applications. In such a case, the business model must provide scenarios that yield to advantages both to the publisher and to the broker. Two cases are then possible:

- If the publisher gets an advantage because the traffic to the publisher’s application can generate revenues for the publisher, then the model should recognize an advantage to the broker for every click to the publisher’s application.
- If instead the publisher provides essential information in order for the application to become possible while having no advantage due to generated traffic, then the situation is opposite, and the model should recognize an advantage to the publisher for every click to the publisher’s application.

Note that a given application might include publishers belonging to both classes. A fair model should then recognize, for every publisher/broker relationship, one of the above cases, and support it through simple contractual conditions. Advertising models can provide the underlying theory for computing the pay-per-click dues.

An interesting aspect is that SeCo applications present as results combinations of individual entries extracted from multiple services, therefore, while clicking on one link, users are choosing a “global solution” which is contributed by all other links offered within the combination. This gives rights to interesting profit sharing schemes that may give advantages also to links that, even if not clicked, contribute to a solution.

Of course, a Search Computing broker publishes a Web application, thus, as any other application, it can host search frames or frames wherein some sponsored links will be impressed, thus using the tools provided by major search companies reviewed in the previous section. Similarly, providers may open frames within their applications, and take advantage of the same mechanisms with the traffic that is carried to it through SeCo applications.

Search computing could then develop its own advertising models, and these models provide interesting problems that we plan to study, initially at a more theoretical level. Two options seem most promising:

- Multi-domain queries in SeCo may offer an important dimension for bidding, by associating bids to keywords only when other specific domains are present in a solution, e.g. a bid for the keyword “movie” only when the user is searching for “renting” in a specific “city”, or instead only when the user is searching for “cinema”. This option could contribute to the current development of flexible auction mechanisms within the scientific community, by adding a relevant dimension.
- A SeCo application could also act as a “broker” for sponsored links, by offering combinations which use them, and by merging the lists of sponsored links returned by multiple service providers in the attempt to rank in high position the

links that are the most appropriate for the users. In such context, the SeCo application could use a model of click probability that takes into account all the click probabilities upon the domains of the query. For instance, the SeCo application could prefer to impress links that are present in the list of all providers, rather than a link that has higher probability in one list but does not appear in the other lists.

These options are currently considered in our research so as to prepare a suitable business and advertising model for Search Computing.

## 6 Conclusions

This chapter aimed at broadening our perspective on Search Computing, from its enabling technologies to its architectural, usage, and business-oriented perspectives. We introduced the roles and tasks of SeCo developers and discussed how design tools can help them. We provided an overview of a reference architecture and deployment strategy, and finally we reviewed advertising models for search industry and thereby introduced the first elements of a business model for SeCo application development.

## References

- [1] Amazon. Elastic Compute Cloud, EC2 (2009), <http://aws.amazon.com/ec2/>
- [2] Hayes, B.: Cloud computing. *Communications of the ACM* 51(7), 9–11 (2008)
- [3] Farrell, J., Nezelek, G.S.: Rich Internet Applications The Next Stage of Application Development. In: 29th International Conference on Information Technology Interfaces, ITI 2007, June 25–28, pp. 413–418 (2007)
- [4] Blumofe, R.D., Leiserson, C.E.: Scheduling multithreaded computations by work stealing. *J. ACM* 46(5), 720–748 (1999)
- [5] Bondi, A.B.: Characteristics of scalability and their impact on performance. In: WOSP 2000: Proceedings of the 2nd international workshop on Software and performance, pp. 195–203. ACM, New York (2000)
- [6] Buyya, R., Abramson, D., Giddy, J., Stockinger, H.: Economic models for resource management and scheduling in grid computing. *Concurrency and Computation: Practice and Experience* 14(13–15), 1507–1542 (2002)
- [7] Buyya, R., Yeo, C.S., Venugopal, S., Broberg, J., Brandic, I.: Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Gener. Comput. Syst.* 25(6), 599–616 (2009)
- [8] Cardellini, V., Casalicchio, E., Colajanni, M., Yu, P.S.: The state of the art in locally distributed web-server systems. *ACM Comput. Surv.* 34(2), 263–311 (2002)
- [9] Craswell, N., Crimmins, F., Hawking, D., Moffat, A.: Performance and cost trade-offs in web search. In: ADC 2004: Proceedings of the 15th Australasian database conference, pp. 161–169. Australian Computer Society, Inc., Darlinghurst (2004)
- [10] Daniel, F., Yu, J., Benatallah, B., Casati, F., Matera, M., Saint-Paul, R.: Understanding UI Integration: A Survey of Problems, Technologies, and Opportunities. *IEEE Internet Computing* 11(3), 59–66 (2007)

- [11] Dean, J., Ghemawat, S.: Mapreduce: simplified data processing on large clusters. In: OSDI 2004: Proceedings of the 6th conference on Symposium on Operating Systems Design & Implementation, pp. 10–10. USENIX Association, Berkeley (2004)
- [12] Even-Dar, E., Kearns, M., Wortman, J.: Sponsored Search with Contexts. In: Deng, X., Graham, F.C. (eds.) WINE 2007. LNCS, vol. 4858, pp. 312–317. Springer, Heidelberg (2007)
- [13] Feng, J., Bhargava, H.K., Pennock, D.: Implementing Sponsored Search in Web Search Engines: Computational Evaluation of Alternative Mechanisms. *Inform Journal on Computing* (forthcoming), <http://ssrn.com/abstract=721262>
- [14] Fielding, R., Gettys, J., Mogul, J.C., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T.: Hypertext transfer protocol (1998), <http://1.1.Tech.rep>.
- [15] Google. AdSense (2009), <https://www.google.com/adsense/>
- [16] Google. AdWords (2009), <https://www.google.com/adwords/>
- [17] Kossmann, D.: The state of the art in distributed query processing. *ACM Comput. Surv.* 32(4), 422–469 (2000)
- [18] Mas-Colell, A., Whinston, M.D., Green, J.R.: *Microeconomic Theory*. Oxford University Press, Oxford (1995)
- [19] Microsoft. Microsoft Advertising (2009), <http://advertising.microsoft.com/>
- [20] Muth, P., Wodtke, D., Weissenfels, J., Dittrich, A.K., Weikum, G.: From centralized workflow specification to distributed workflow execution. *J. Intell. Inf. Syst.* 10(2), 159–184 (1998)
- [21] Narahari, Y., Garg, D., Narayanam, R., Prakash, H.: *Game theoretic problems in network economics and mechanism design solutions*. Springer, Berlin (2009)
- [22] Pfister, G.F.: *In search of clusters*, 2nd edn. Prentice-Hall, Inc., Upper Saddle River (1998)
- [23] Shafer, J.C., Agrawal, R., Lauw, H.W.: *Symphony: Enabling Search-Driven Applications*. In: USETIM (Using Search Engine Technology for Information Management) Workshop, VLDB Lyon (2009)
- [24] Weber, T.A., Zheng, Z.E.: A model of search intermediaries and paid referrals. *Tech. rep.*, 02-12-01, The Wharton School, University of Pennsylvania (2003), [http://papers.ssrn.com/sol3/papers.cfm?abstract\\_id=601903](http://papers.ssrn.com/sol3/papers.cfm?abstract_id=601903)
- [25] Yahoo! APT from Yahoo! (2009), <http://apt.yahoo.com/>
- [26] Yahoo! SearchMarketing (2009), <http://searchmarketing.yahoo.com/>
- [27] Yang, H.C., Dasdan, A., Hsiao, R.L., Parker, D.S.: Map-reduce-merge: simplified relational data processing on large clusters. In: SIGMOD 2007, pp. 1029–1040. ACM, New York (2007)



# Chapter 15:

## Search Computing and the Life Sciences

Marco Masseroli<sup>1</sup>, Norman W. Paton<sup>2</sup>, and Irena Spasić<sup>2</sup>

<sup>1</sup> Dipartimento di Elettronica e Informazione, Politecnico di Milano,  
Piazza Leonardo da Vinci 32, 20133 Milano, Italy  
masseroli@elet.polimi.it

<sup>2</sup> School of Computer Science, University of Manchester,  
Oxford Road, Manchester M13 9PL, UK  
{npaton, i.spasic}@manchester.ac.uk

**Abstract.** Search Computing has been proposed to support the integration of the results of search engines with other data and computational resources. A key feature of the resulting integration platform is direct support for multi-domain ordered data, reflecting the fact that search engines produce ranked outputs, which should be taken into account when the results of several requests are combined. In the life sciences, there are many different types of ranked data. For example, ranked data may represent many different phenomena, including physical ordering within a genome, algorithmically assigned scores that represent levels of sequence similarity, and experimentally measured values such as expression levels. This chapter explores the extent to which the search computing functionalities designed for use with search engine results may be applicable for different forms of ranked data that are encountered when carrying out data integration in the life sciences. This is done by classifying different types of ranked data in the life sciences, providing examples of different types of ranking and ranking integration needs in the life sciences, identifying issues in the integration of such ranked data, and discussing techniques for drawing conclusions from diverse rankings.

**Keywords:** search computing, bioinformatics, data integration, ranked data.

### 1 Introduction and Motivation

Experimental studies in the life sciences give rise to **large quantities of diverse, complex data**. Taking *genomics* as an example, initial sequencing work gives rise to a raw genome sequence, which in turn is annotated with the predicted locations of genes. In essence, every cell in an organism contains the same genome sequence, but biological processes cause the products described by the genome to be created differently in different cells. For example, different cells contain different collections of proteins, and over time the quantities of different proteins within a cell vary. As a result, to understand the dynamic behavior of a cell, it is necessary to measure quantities of different types of molecules within the cell. *Functional genomics* encompasses a collection of experimental techniques, including transcriptomics, proteomics and metabolomics, which are used to measure the quantities of mRNA,

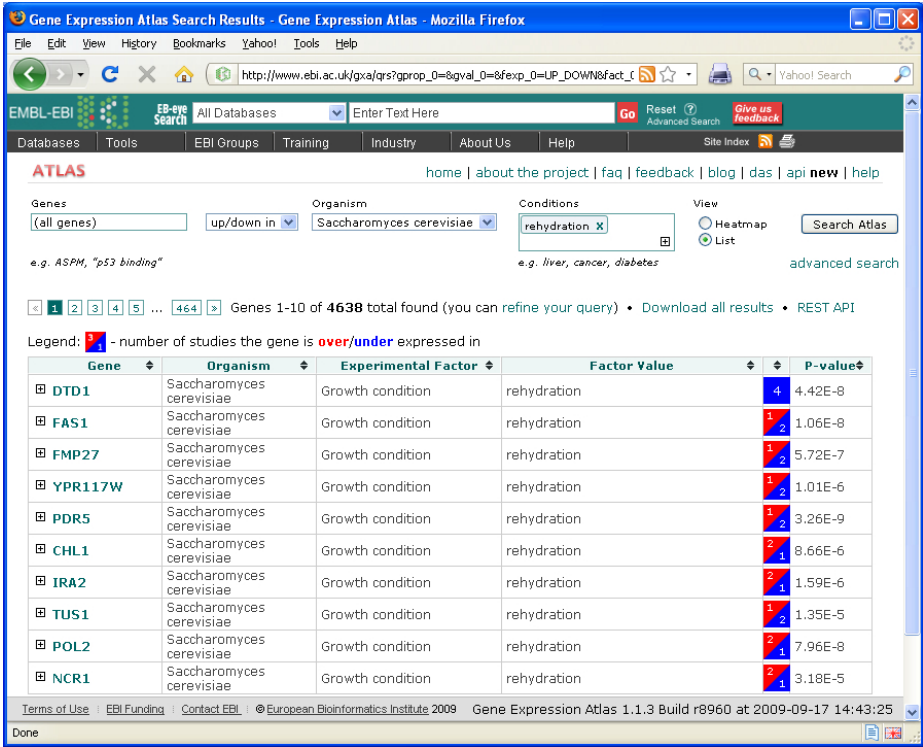


Fig. 1. Gene expression data result from ArrayExpress

protein and small molecules associated with a cell at runtime. In all forms of functional genomics, initial experimental readings are processed by software to produce derived results, which represent the conclusions of the experiment. All experimental methods involve some level of uncertainty in the results produced, and in functional genomics the uncertainty may result either from the experimental method used or from the analyses performed on the raw data (e.g. [1]). Both numerical experimental results and the uncertainties associated with measurements may give rise to **rankings** in experimental data sets. An example is given in Figure 1 from the ArrayExpress microarray database [2], which indicates the number of experiments carried out on *Saccharomyces cerevisiae* (yeast) where the growth condition is *rehydration*, in which specific genes have been found to be up or down regulated, along with an associated *P-value*. This data set could usefully be ordered either by the number of examples of up/down regulation or the associated *P-value*.

The diversity of organisms studied and types of experimental data have given rise to a **proliferation of data resources**, and as a result, data management and integration are high profile activities in the life sciences. The **resources** are also **diverse in their nature**; in addition to sequence and functional genomics data resources available in structured or semi-structured repositories (generally heterogeneous and distributed) [3], much of the accrued scientific knowledge has been published in the scientific literature. So it is often important either to be able to

retrieve documents that describe some experimentally identified phenomenon, or to be able to extract specific values from the literature using information extraction techniques [4] and [5]. As such, **multi-domain data integration** is central to the life sciences, and may involve the combination of search with other data access and analysis tasks, as envisaged in Search Computing [6]. In essence, search computing seeks to support declarative expression of requests over multiple search and data services, where the search services can be applied to multiple domains and are characterized by incremental production of potentially huge, ranked answers. The presence of search services as part of the integration process means that ranking, and the combination of multiple rankings, also coming from different domains or being of different nature, needs to be accommodated during data integration. In practice, data integration in the life sciences has used many different techniques [7], including warehousing (e.g. [8]), workflows (e.g. [9]) and distributed query processing (e.g. [10]). However, the infrastructures that support such integration rarely provide direct and/or transparent support for ordered data, which means that where ordering is considered, this must either form part of the integration application, or result from the use of analysis techniques that take ordering into account. Thus, ordering is rarely supported as a first class citizen either for individual data resources, or in information integration platforms that act over multiple domains.

As ordered data may originate from several sources, it may be appropriate for a final ordering of an integration task to be computed based on properties of the contributing results. In the web context, *rank aggregation* [11] has been investigated with a view to developing algorithms that combine multiple search engine rankings while remaining computationally tractable. However, **different sorts of evidence** may be available in different sources. For example, in addition to the gene expression data from ArrayExpress illustrated in Figure 1, there may be further gene expression evidence from GEO [12] and information on protein expression from PRIDE [13], and it may be appropriate to rank the overall result of a multi-domain search for up-regulated genes based on evidence from all three sources. However, most proteomics experiments captured in PRIDE are qualitative, and thus principally provide presence/absence information, whereas gene expression experiments typically provide quantitative information. Interpreting this quantitative information is complicated, however, by the fact that, although some gene expression experiments provide absolute measures, most provide relative measures. As a result, computing an appropriate overall ranking from information derived from a collection of heterogeneous sources may not be straightforward.

The following sections in this chapter explore the origin, nature and role of ordered data in the life sciences, and in particular its relevance to information integration. Section 2 focuses on ordered data in the life sciences, characterizing its properties, and providing examples of such data. Section 3 discusses the integration of ordered data sets, where the integration takes account of the different forms of order. Section 4 considers the different approaches existing to manage the integration of different types of ordered data, and the lack of explicit support to manage rankings in current integration platforms. Section 5 presents some examples of life-science specific case studies where ranking of the data to be integrated matters. Some conclusions on the relationship between search computing and the life sciences are presented in Section 6. In so doing, we hope both to present some new challenges to

search computing where the initial emphasis has been on ranked data produced by multi-domain search engines, and to encourage wider exploration of middleware support for ordered data in the life sciences.

## 2 Ordered Data in the Life Sciences

In this section we describe the basic concepts related to the notion of ordering and demonstrate how they apply to data in the life sciences.

A basic concept upon which the notion of order is founded is a binary relation, which can be intuitively interpreted as “less than”, and consequently can be used to say that one object precedes another one, hence the intuitive notion of ordering. We differentiate between different types of order depending on the properties of the underlying binary relation  $R$  over a set  $S$ , i.e.  $R \subseteq S \times S$ . A relation  $R$  is *reflexive* if  $(x, x) \in R$  for all  $x \in S$ . It is *antisymmetric* if  $(x, y) \in R$  and  $(y, x) \in R$  implies that  $x = y$  for all  $x, y \in S$ . Finally, a relation  $R$  is *transitive* if  $(x, z) \in R$  whenever  $(x, y) \in R$  and  $(y, z) \in R$  given any  $x, y, z \in S$ . A binary relation that is both *reflexive* and *transitive* defines a *quasi-order*. A quasi-order relation that is also *antisymmetric* defines a *partial order*, and is usually denoted with ‘ $\leq$ ’. Thus, a partial order implies that all the following conditions are true for all elements  $x, y, z$  of a set  $S$ : (i)  $x \leq x$ ; (ii) if  $x \leq y$  and  $y \leq x$ , then  $x = y$ ; and (iii) if  $x \leq y$  and  $y \leq z$ , then  $x \leq z$ . In a set with a partial order (a *partially ordered set*, or *poset*) it may not always be possible to say which of two elements “precedes” the other. This gives rise to a *total order*, i.e. a partial order that satisfies the *totality* condition, in which any two elements  $x$  and  $y$  can be related to each other as  $x \leq y$  or  $y \leq x$ . A set with a total order is a *totally ordered set*.

In addition to partially and totally ordered sets, it is important to formalize the sense of an object being “strictly less than” another. This gives rise to a *strict partial order*, a binary relation that is *irreflexive* and *transitive*. It is usually denoted with ‘ $<$ ’. A strict partial order is called a *strict total order* if it satisfies the *trichotomy* condition, i.e. for all elements  $x$  and  $y$  it is  $x < y$  or  $y < x$  or  $x = y$ . It is important to emphasize that a strict partial order is not a type of partial order. Indeed it cannot be a partial order, since it is not reflexive. Finally, an order may be induced on a set  $S$  with a real-valued function  $f: S \rightarrow \mathbf{R}$  that satisfies the following condition for all elements  $x, y \in S$ :  $x \leq y$  if and only if  $f(x) \leq f(y)$ .

An important data structure related to partial orders is that of a directed acyclic graph (or DAG), which is a directed graph with no directed cycles, i.e. no node in such a graph can be returned to by following the links between the nodes. The reachability relation in a DAG is a partial order on the set of its nodes, and conversely any finite partial order may be represented by a DAG. In the life sciences, DAGs together with trees and sequences as their special cases are used to represent taxonomies of proteins, chemical compounds and organisms, paronomies, data provenance, multiple sequence alignments, gene clustering, sequence data (e.g. DNA sequences), etc. [14]. However, the nature of some life science phenomena is intrinsically cyclic and their representation requires more complex structures such as a directed cyclic graph, a natural extension of a DAG, in which cycles are allowed. They can be used to model a wide range of biochemical processes in which materials

or signals flow through a network of nodes, e.g. metabolic pathways, signaling pathways, gene regulatory networks, etc.

Apart from the examples of **order found in nature**, much of the data produced or used in life science experiments can be related by way of partial orders. More precisely, such an order is often **induced by measuring certain physical properties**. The ordering may be used both as input and/or output of biochemical experiments. Take for an example a study of the effects of growth rate on the levels of gene expression, proteins and metabolites in the yeast, *Saccharomyces cerevisiae* [15]. Each data set was generated by analysing yeast grown under different limiting conditions involving different nutrients, each used for three sub-experiments with a different dilution rate (0.07, 0.10 and 0.20) for the given nutrient. Therefore, samples can be partially ordered by the input dilution rate. Similarly, the experimental results can be ordered using the output values such as metabolite concentrations obtained by analysing the samples using gas chromatography–mass spectrometry.

In addition to using the measurements of physical properties, the objects of life science experiments can be **ordered by using scores produced by statistical and/or computational analysis of the associated data**. The similarities between either nucleotide or protein sequences are used to infer gene function, new members of gene families and evolutionary relationships. Basic Local Alignment Search Tool (BLAST) is frequently used to estimate sequence similarity [16]. BLAST combines a bit score, which takes into account the alignment of similar or identical residues, as well as any gaps introduced to align the sequences, with expected value (E-value) as an indication of the statistical significance of a given pairwise alignment. The BLAST score is used to order similar biomolecular sequences, ideally in a way that represents the closeness of their evolutionary history.

Another important source of order in the life sciences is the literature, which is the prevalent medium for information exchange among experts in the field. The rapidly expanding volume of the life science literature makes it difficult to efficiently locate, retrieve and manage relevant information. One of the basic concepts used to facilitate access to documents relevant for the given search terms is that of TF–IDF (term frequency – inverse document frequency), a statistical measure that estimates the importance of a word relative to a document [17]. Its value is directly proportional to the frequency of a word within the document and indirectly proportional to its frequency within the considered collection of documents. Given a search term, TF–IDF is used to induce an order over a set of documents (see Figure 2 for an example). Text annotations (e.g. genes, species, protein-protein interactions, etc.) provided automatically by different information extraction systems together with their confidence scores represent another source of ordering documents based on their semantic relevance [18].

To sum up, order – in particular a partial order – is an intrinsic characteristic of many phenomena researched in life sciences, as well as a useful way of organizing our knowledge about them. The phenomena themselves can typically be ordered in two basic ways: spatial and temporal. For example, genes are ordered in spatial terms within the genome, whereas metabolic reactions occur in a given temporal order, which is reflected in the graph structures used to represent metabolic pathways. When biological objects cannot be ordered directly in such a manner, their quantitative properties can often be used as a basis to induce their ordering. These properties can

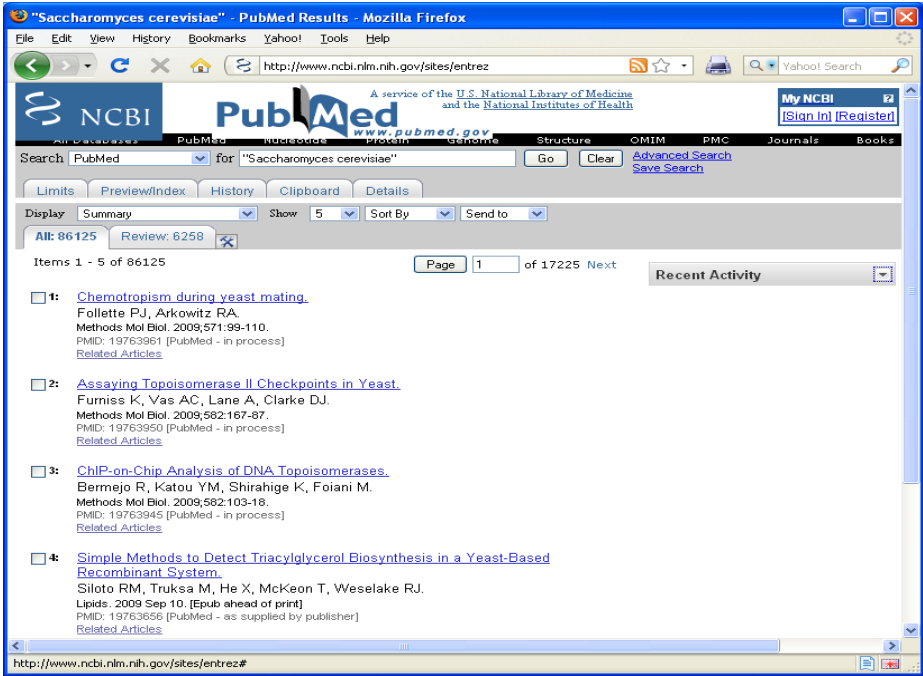
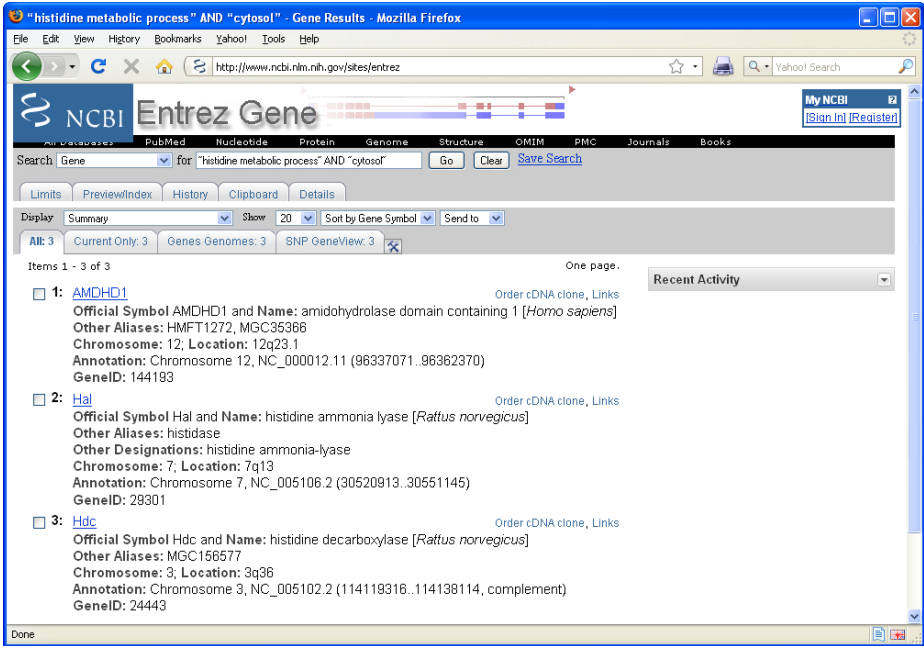


Fig. 2. PubMed search result for “*Saccharomyces cerevisiae*”

be the measurements of conditions applied in an experimental protocol used to produce the physical objects researched (e.g. dilution rates in the growth medium), or they can be the actual values measured by the experiment (e.g. metabolite concentrations). In addition, the scores calculated by bioinformatics algorithms can be used as a source of order in the same way. Finally, order is often used in modeling the knowledge accumulated in the life sciences, and these structures usually reflect the intrinsic ordering among the phenomena being modeled. For example, phylogenetic trees are the main tool for representing evolutionary relationship among species, which itself is a partial order.

### 3 Combining Multi-domain Ordered Data in the Life Sciences

In recent years, a lot of work has been done to ease management and access to the ever increasing amount of genomic and proteomic data and knowledge available. A significant number of web interfaces and services [19] are publicly available for exploring and searching repositories containing information about biomolecular entities (i.e. DNA sequences, genes, transcripts and proteins) or structural, functional and phenotypic biological features (e.g. sequence polymorphisms, biological processes, biochemical pathways or genetic disorders), preferably expressed through controlled terminologies and ontologies, as well as the associations of the former with the latter (Figure 3). Much effort has also been devoted to the interlinking and



**Fig. 3.** Search result from Entrez Gene databank. All genes associated with “*histidine metabolic process*” in “*cytosol*” are retrieved.

integration of such data, mainly to support navigation through the many repositories in which data relevant to a search may be sparsely stored, or to aggregate data for further analysis [20]. In both cases, the focus has mainly been on coverage, with the aim of aggregating as much data as possible, usually without considering possible rankings of the data to be integrated. This approach is adequate when the data to be integrated are limited in quantity, or a holistic result is required. Yet, when the data are voluminous and their low partial ranking indicates low relevance or even possibly erroneous data, considering only the top-*k* data items may improve integration performance and quality of global results.

Search computing techniques have as a key feature the ability to compose multi-domain ranked partial results from single-domain searches. Thus, it appears interesting to consider their **application in searching for answers to life science questions**, which can be addressed only by comprehensively analyzing different types of data that are inherently ordered, or are associated with ranked confidence values. Examples of life science questions that may be answered with such searches are: “Which are the proteins in different organisms that are more structurally and functionally similar to a given protein?”, “Which are the genes that have the highest sequence similarity in different model organisms and are highly co-expressed in the same biological conditions?”, “Which are the proteins encoded by co-expressed genes that are more likely to interact?”, “Which are the drugs available to treat the diseases known to be more likely associated with a given genetic mutation?”, or “Which are the highest risk factors associated with the most prevalent diseases among young

people?”. By using available web services for searching biomolecular data, and taking advantage of the attributes they define for providing a ranking, search computing techniques should prove applicable in the life sciences. That is, they may offer support for the goal of providing **online integrated and prioritized results of complex multi-domain searches**, e.g. aiming to search for evidence of correlations between information of different domains (e.g. genotype-phenotype correlations).

Difficulties, however, may arise from the nature of the life science data and from the different features of order of the diverse domains to be integrated. For example, when ranking of the data to be integrated is defined by probability values (representing data confidence, evidence, or correctness), it is reasonable to evaluate whether ranking composition might be the best option, or if the probabilistic combination of their probability values could provide better overall ordered integration. Furthermore, ranking composition strategies currently proposed in search computing penalize the global ranking of items with missing values in any of the integrated domains. This aspect may have a negative impact on the final results when several domain data subsets are integrated and missing values do not necessarily have a negative connotation. Both aspects hold for some life science data. In particular, missing values may merely indicate “unavailable” information, and in some cases may imply items of particular interest, which may require more in depth analyses. Specific heuristics, or configuration parameters, could therefore be required for search computing methods to deal with such issues effectively.

## 4 Managing Domain Ranking in Integrated Data

In current integration platforms, usually no explicit support is available to manage rankings encountered in the different domains of the data to be integrated, which at best is reflected as additional data attribute(s) available for further processing. Thus, domain ranking needs to be handled in the applications built on top of the integrated data.

A range of approaches exists to manage the integration of different types of ranked data. Among them, *rank aggregation* techniques are the most general ones. Rank aggregation, broadly discussed in Chapter 11 of this book, has the ability to combine ordered lists coming from different sources and platforms as one of its major strengths. In the life science context, it has been proposed for data clustering [21] and for the meta-analysis of different microarray gene expression studies from heterogeneous platforms, which may not necessarily be directly comparable otherwise. For example in [22] a meta-analysis of 20 studies on multiple cancers performed with different microarray chips has been carried out using rank aggregation algorithms. Since the final results of microarray studies are typically expressed as lists of genes rank-ordered by a measure of the strength of evidence that they are functionally involved in the biological process under investigation, this approach allows the rank-order to abstract from the actual expression levels, which may not be comparable across experiments.

Lately, a specific package for weighted rank aggregation [23] has been developed for R, an open source statistical program widely used in bioinformatics and computational biology. It implements and makes easily usable some rank aggregation algorithms, including *Borda count*, *Cross-Entropy Monte Carlo algorithm*, *Genetic algorithm*, and a *brute-force algorithm* (for small problems). In particular, the Cross-Entropy Monte Carlo



algorithm is an iterative procedure for solving difficult combinatorial problems in which it is not computationally feasible to find the solution directly.

As illustrated in Chapter 11 of this book, other approaches to handling the integration of diverse ranked data exist. They include *Top-k* and *Skyline* techniques [24] and [25], and also *probabilistic rank aggregation* methods [26] and [27]. *Top-k* ranks the top  $k$  tuples in terms of a user-defined score function, while *Skyline* identifies non-dominated tuples, i.e. the tuples such that no other tuple is better against all user criteria. Probabilistic rank aggregation methods use Machine Learning and Information Retrieval approaches to estimate the posterior distribution of the target rank. However, *Top-k*, *Skyline* and probabilistic rank aggregation approaches are not especially prevalent in the life sciences, possibly because some of them require restrictive assumptions that are rarely satisfied in practical life science cases that, for example, frequently involve incomplete data.

Taking into account different domain rankings simultaneously is also equivalent to considering the levels of evidence during their integration in support of some global evaluation of the data. This is a typical multi-optimization problem in the presence of possibly conflicting objectives. To approach and solve such problems, some computational techniques exist. The main one is *multi-objective optimization*, also known as multi-criteria or multi-attribute optimization, which allows simultaneous optimizing of multiple objectives that are subject to certain constraints [28] and [28]. Examples of multi-objective optimization problems in the life sciences include finding proteins with the highest confidence of interaction coded for by genes with the highest co-regulation, or finding in different model organisms genes with the highest sequence similarity and the highest expression levels in the same biological conditions. For well formed multi-objective problems, no single solution exists that simultaneously optimizes each single objective to its fullest. The solution is the one for which each objective is optimized to the extent that optimizing it any further causes the other objective(s) to worsen. Finding such a solution, and quantifying how much better it is than other such solutions (that are generally numerous) is the goal when setting up and solving a multi-objective optimization problem.

The solution of a multi-objective problem is a (possibly infinite) set of Pareto points. Pareto solutions are those for which improvement in one objective can only occur with the worsening of at least one other objective. The most intuitive approach to solving the multi-objective problem is constructing a single aggregate objective function. The basic idea is to combine all objective functions into a single functional form. A well-known combination is the weighted linear sum of the objectives. It consists of specifying scalar weights for each objective to be optimized, and then combining them into a single function that can be solved by any single-objective optimizer. Yet, the solution obtained will depend on the values (more precisely, the relative values) of the weights specified. Thus, the weighted sum method is essentially subjective, in that often there is no objective way to define weights univocally. Moreover, this approach cannot identify all Pareto solutions.

The objective way of solving multi-objective problems requires a Pareto-compliant ranking method, favoring Pareto solutions, as in current multi-objective evolutionary approaches [30]. They do not require weights, and thus no *a priori* information on the problem is needed. Furthermore, most evolutionary optimizers apply Pareto-based ranking schemes. *Genetic algorithms* such as the Non-dominated Sorting Genetic

Algorithm-II (NSGA-II) [31] and Strength Pareto Evolutionary Approach 2 (SPEA-2) [32] have become standard approaches in multi-objective optimization, although some schemes based on particle swarm optimization and simulated annealing are significant.

Some examples of genetic algorithm optimization in the life sciences can be found in [33]. The most significant applications are for classification and inverse problems. The latter ones arise where the data generated by a biological process or system can be measured and the aim is to reconstruct the original system from the observed data, which can be noisy and of several types. Many such cases exist in the life sciences, e.g. in evolutionary biology the inference of phylogenetic trees from biological sequence alignment data, or in functional genomics the inference of gene regulatory networks (GRN) from gene expression data. GRN are networks of inhibitory and stimulatory interactions between genes that model the complex interplay mechanisms of interactions between DNA, RNA, proteins and metabolites, which determine different patterns of gene expression. The optimization of classification problems regards the performance optimization of classifiers (e.g. for the distinction between tumor and healthy patients) by optimizing the trade-off between their conflicting performance measures of sensitivity and specificity (i.e. by minimizing both false negatives and false positives). It also addresses feature selection, by identifying the minimum set of features that give the best classification accuracy, mainly using unsupervised classification (i.e. clustering), in particular for gene expression raw data analysis. In clustering, the multi-objective approach has been used for direct optimization of the clustering partitioning with respect to a number of complementary clustering criteria giving different clustering results, and for the selection of the best number of clusters.

Other life science applications in which multi-objective optimization has been investigated include inference of protein networks and metabolic pathways from experimental data; assessment of sequential and structural similarities of DNA and RNA macromolecules, as well as proteins; identification of motifs; protein structure prediction; and selection of single-nucleotide polymorphisms. In most cases, however, multi-objective optimization is applied on single source data, or on homogenous data from multiple sources. Rare examples of multi-domain data integration exist for multi-objective approaches, whose full potential in comparison to the current state-of-the-art techniques remains to be explored.

## 5 Data Integration Case Studies in Which Order Matters

In the life sciences several data integration examples exist in which the order of the data to be integrated is relevant. Some case studies are presented below.

### 5.1 Identifying Genes Relevant to a Disease

Genetically inherited diseases are linked to a region of a human genome, but the association between a disease known to be genetically inherited and a particular gene remains unknown for hundreds of such diseases. The online availability of the information about the human genome, both in a structured form (e.g. biomolecular sequence databases) and described in the literature, has prompted the development of computational approaches to generating hypothesis about possible gene-disease links.

However, no single source would suffice to provide enough evidence for such associations. Therefore, this area of biomedical research requires integration of different types of data ranked using different criteria. We can imagine one plausible scenario for the application of search computing for this particular problem.

Perez-Iratxeta, *et al.* [34] developed a data-mining system, based on fuzzy set theory, which performs the prioritization of candidate genes for genetically inherited diseases for which no underlying gene has yet been assigned. Their approach evaluates and scores associations between gene functions and disease phenotypes by combining the information from MEDLINE and a protein sequence database. Given a disease, this score induces a partial order on the set of potentially related genes. Further, one may wish to explore relationships between the genes themselves. This can be done by using BLAST to estimate their sequence similarity [16]. Alternatively, one could also use the literature to retrieve biological relationships between genes according to co-occurrence based meta-analysis of scientific literature [35]. The associative concept space (ACS) has been developed for the representation of information extracted from biomedical literature as a Euclidean space in which thesaurus concepts are positioned and the distances between concepts indicates their relatedness. Given a gene, the distance in this space can be used to rank other potentially related genes. Finally, genes can be explored using high-density microarray technology. A great variety of statistical approaches have been applied to identify clusters of genes that share common expression characteristics [36]. For example, in case of partitive clustering where the data are divided using a similarity measure, this measure can be used to order the genes based on their estimated similarity. When it comes to agglomerative techniques, the hierarchical cluster structure again provides a partial order between the genes based on the reachability relation between the nodes in the hierarchy.

Obviously, answering the questions in this important area of genetic research requires careful integration of data originating from different sources and ranked using different criteria. Automating the searches over these sources, together with the integration of the ranked search results, can provide valuable clues about the links between diseases and the genes related to them. Therefore, this would be a natural application area for search computing within the life sciences.

## 5.2 Identifying Genes Associated with Traits in Quantitative Trait Loci

Quantitative Trait Loci (QTLs) are regions of DNA associated with a complex phenotype that varies by degree, such as height or susceptibility to polygenic diseases such as diabetes or cancer [37]. A QTL region may contain many genes, so, given a collection of QTL regions associated with some traits, an important question is which genes contribute directly to a trait. One approach to identifying genes of relevance to a trait is to carry out microarray experiments to establish which genes are differentially expressed for individuals with different phenotypes. For example, Datta *et al.*, [38] use microarrays to identify genes that are expressed differently in strains of mice that vary in their abilities to expel a parasite. In this setting, there are several different potential sources of partially ordered data. For example, different genes may be more central to a QTL region than others, and some genes may have different levels of expression change or different probabilities that their expression has changed

in a significant way. In this context, it is certainly of interest to identify genes for which there is the greatest evidence that they are associated with a given trait, but the rank orders of the contributing data sets are unlikely to be the most informative criteria to use for identifying a globally partial order.

### 5.3 Comparative Genomic Analysis

Comparative genomics is the study of the relationship of genome structure and function across different biological species. It attempts to take advantage of both similarities and differences in proteins, RNA, and regulatory regions of different organisms to understand the function and evolutionary processes that act on genomes. The main analysis performed in comparative genomic studies is the alignment of biomolecular sequences (nucleotide or amino acidic) of different organisms to search for their degree of similarity (i.e. homology). Although several different algorithms exist to perform such analysis, BLAST [16] is the most widely used. BLAST searches for the nucleotide or amino acidic sequences most similar to a given query sequence by comparing it with a database of sequences, and identifying the sequences that resemble the query sequence above a certain threshold. For example, following the discovery of a previously unknown gene in the mouse, a scientist typically performs a BLAST search of the human genome to see if humans carry a similar gene; BLAST identifies sequences in the human genome that resemble the mouse gene based on similarity of sequence. The output of a BLAST search is a list of similar sequences ordered by their degree of similarity to the query sequence (Figure 4). It therefore constitutes a partially ordered set of similar sequences, since two sequences in the set can have the same degree of similarity to the query sequence.

Alignment	DB:ID	Source	Length	Score	Identity%	Positives%	E()
1 <input type="checkbox"/>	<a href="#">EM_PAT:DD130059</a>	Diagnosis and Prognosis of Breast Cancer Patients.	1992	9960	100	100	0.
2 <input type="checkbox"/>	<a href="#">EM_PAT:DD208683</a>	Expression Profile of Prostate Cancer.	1992	9960	100	100	0.
3 <input type="checkbox"/>	<a href="#">EM_PAT:DD415310</a>	Diagnosis and Prognosis of Breast Cancer Patients.	1992	9960	100	100	0.
4 <input type="checkbox"/>	<a href="#">EM_PAT:GM974767</a>	Sequence 120 from Patent EP2003213.	1992	9960	100	100	0.
5 <input type="checkbox"/>	<a href="#">EM_PAT:AR274918</a>	Sequence 55 from patent US 6506607.	1992	9960	100	100	0.
6 <input type="checkbox"/>	<a href="#">EM_PAT:EA062820</a>	Sequence 645 from patent US 7171311.	1992	9960	100	100	0.
7 <input type="checkbox"/>	<a href="#">EM_PAT:EA248485</a>	Sequence 120 from patent US 7229774.	1992	9960	100	100	0.
8 <input type="checkbox"/>	<a href="#">EM_PAT:EA427947</a>	Sequence 120 from patent US 7332290.	1992	9960	100	100	0.
9 <input type="checkbox"/>	<a href="#">EM_PAT:GP320972</a>	Sequence 645 from patent US 7514209.	1992	9960	100	100	0.
10 <input type="checkbox"/>	<a href="#">EM_HUM:M27396</a>	Human asparagine synthetase mRNA, complete cds.	1992	9960	100	100	0.
11 <input type="checkbox"/>	<a href="#">EM_PAT:CQ875273</a>	Sequence 16 from Patent WO2004076613.	1994	9895	99	99	0.
12 <input type="checkbox"/>	<a href="#">EM_PAT:CS063065</a>	Sequence 49 from Patent EP1522594.	1994	9895	99	99	0.
13 <input type="checkbox"/>	<a href="#">EM_PAT:CS080846</a>	Sequence 49 from Patent WO2005040414.	1994	9895	99	99	0.
14 <input type="checkbox"/>	<a href="#">EM_PAT:DD387278</a>	COMPOSITIONS AND METHODS FOR THE DIAGNOSIS AND TREATMENT OF TUMOR.	1994	9895	99	99	0.
15 <input type="checkbox"/>	<a href="#">EM_PAT:DL464877</a>	COMPOSITIONS, KITS, AND METHODS FOR IDENTIFICATION, ASSESSMENT, PREVENTION, AND THERAPY OF CANCER.	1994	9895	99	99	0.
16 <input type="checkbox"/>	<a href="#">EM_PAT:FB671589</a>	Sequence 49 from Patent EP1892306.	1994	9895	99	99	0.

**Fig. 4.** Example of ordered BLAST search result for the nucleotide sequence “*Human asparagine synthetase mRNA*”

In such a comparative genomic context, an important task is to find the genes in different organisms that are most structurally and functionally similar to a given gene. To address this aim, it is thus possible to run a BLAST search for that gene and then to search (e.g. in ArrayExpress) for expression data for that gene and for the gene sequences obtained by the BLAST search. Integration of the search results by looking for the gene sequences co-expressed with the given gene in analogous biological conditions can provide the sought genes. Since gene expression results are intrinsically ordered (see Figure 1) and constitute a partially ordered set, the order of the BLAST and the expression data search results to be composed contribute to identifying the most structurally and functionally similar genes. Requiring multi-domain search and composition of the obtained ranked search results, this would therefore be a natural application area for search computing within the life sciences.

## 6 Conclusions

This chapter has explored the extent to which the proposed search computing functionalities, which feature multi-domain composition of ranked partial results from single-domain searches, are able to help answer life science questions that require the integration of different data types (domains) and forms of ranked data. In particular, this chapter has investigated ordered data in the life sciences, with a view to understanding the challenges such data present to information integration platforms, such as those proposed by search computing. The following observations can be made:

1. Ordered data, and in particular partially ordered data, are extremely prevalent in the life sciences, and are poorly served by current data integration platforms.
2. Ordered data are highly heterogeneous, in that: (i) ordered data may describe a wide variety of physical, experimental and analytical features; and (ii) the ordering may represent a range of different notions, such as quantity, confidence, or location.

As such, providing support for ordering as a first class citizen in integration platforms in the life sciences seems appropriate. However, it seems likely that no single mechanism for aggregating ordered data sets will meet the diversity of user requirements. Nevertheless, supporting multi-domain integration of ordered data should add value to the results of integration tasks by reducing the need for *ad hoc* post processing, and may increase the complexity of life science questions that integration tools can support directly. For example, questions that seem likely to be able to be addressed using search computing techniques include: “Which proteins in different organisms are most structurally and functionally similar to a given protein?”, “Which genes have the highest sequence similarity in different model organisms and are highly co-expressed in the same biological conditions?”, “Which proteins are encoded by co-expressed genes that are likely to interact?”, and “Which drugs threat diseases that are likely to be associated with a given genetic mutation?”.

Search computing techniques may therefore help in selecting and prioritizing life science search results according to their collected multi-domain characteristics and

the associated confidence values. However, the complex nature of life science data, the frequency of unavailable or missing values, the diversity of ordering types and the challenges of combining those orderings present challenges that require further investigation. Such challenging life science applications may nonetheless represent a good test bed for advanced search computing applications, with valuable spin-offs for other scenarios.

## References

1. Stead, D., Paton, N.W., Missier, P., Embury, S.M., Hedeler, C., Jin, B., Brown, A.J.P., Preece, A.D.: Information quality in proteomics. *Brief. Bioinform.* 9(2), 174–188 (2008)
2. Parkinson, H., Sarkans, U., Shojatalab, M., Abeygunawardena, N., Contrino, S., Coulson, R., Farne, A., Lara, G.G., Holloway, E., Kapushesky, M., Lilja, P., Mukherjee, G., Oezcimen, A., Rayner, T., Rocca-Serra, P., Sharma, A., Sansone, S., Brazma, A.: ArrayExpress—a public repository for microarray gene expression data at the EBI. *Nucleic Acids Res.* 33(Database issue), D553–D555 (2005)
3. Galperin, M.Y., Cochrane, G.R.: Nucleic Acids Research annual database issue and the NAR online molecular biology database collection in 2009. *Nucleic Acids Res.* 37(Database issue), D1–D4 (2009)
4. Krallinger, M., Valencia, A., Hirschman, L.: Linking genes to literature: text mining, information extraction, and retrieval applications for biology. *Genome Biol.* 9(suppl. 2), S8 (2008)
5. Spasic, I., Ananiadou, S., McNaught, J., Kumar, A.: Text mining and ontologies in biomedicine: making sense of raw text. *Brief. Bioinform.* 6(3), 239–251 (2005)
6. Braga, D., Ceri, S., Daniel, F., Martinenghi, D.: Mashing up search services. *IEEE Internet Comput.* 12(5), 16–23 (2008)
7. Hernandez, T., Kambhampati, S.: Integration of biological sources: current systems and challenges ahead. *SIGMOD Record* 33(3), 51–60 (2004)
8. Masseroli, M., Ceri, S., Campi, A.: Integration and mining of genomic annotations: experiences and perspectives in GFINDER data warehousing. In: Paton, N.W., Missier, P., Hedeler, C. (eds.) *DILS 2009. LNCS (LNBI)*, vol. 5647, pp. 88–95. Springer, Heidelberg (2009)
9. Hull, D., Wolstencroft, K., Stevens, R., Goble, C.A., Pocock, M.R., Li, P., Oinn, T.: Taverna: a tool for building and running workflows of services. *Nucleic Acids Res.* 34, 729–732 (2006)
10. Goble, C.A., Stevens, R., Ng, G., Bechhofer, S., Paton, N.W., Baker, P.G., Peim, M., Brass, A.: Transparent access to multiple bioinformatics information sources. *IBM Systems Journal* 40(2), 534–551 (2001)
11. Dwork, C., Kumar, R., Naor, M., Sivakumar, D.: Rank aggregation methods for the web. In: *Proceedings of the 10th International World Wide Web Conference, WWW 2001*, pp. 613–622. ACM Press, New York (2001)
12. Edgar, R., Domravec, M., Lash, A.E.: Gene Expression Omnibus: NCBI gene expression and hybridization array data repository. *Nucleic Acids Res.* 30(1), 207–210 (2002)
13. Jones, P., Côté, R.G., Martens, L., Quinn, A.F., Taylor, C.F., Derache, W., Hermjakob, H., Apweiler, R.: PRIDE: a public repository of protein and peptide identifications for the proteomics community. *Nucleic Acids Res.* 34(Database Issue), D659–D663 (2006)
14. Olken, F.: Graph data management for molecular biology. *OMICS: A Journal of Integr. Biol.* 7(1), 75–78 (2003)

15. Castrillo, J.I., Zeef, L.A., Hoyle, D.C., Zhang, N., Hayes, A., Gardner, D.C., Cornell, M.J., Petty, J., Hakes, L., Wardleworth, L., Rash, B., Brown, M., Dunn, W.B., Broadhurst, D., O'Donoghue, K., Hester, S.S., Dunkley, T.P., Hart, S.R., Swainston, N., Li, P., Gaskell, S.J., Paton, N.W., Lilley, K.S., Kell, D.B., Oliver, S.G.: Growth control of the eukaryote cell: a systems biology study in yeast. *J. Biol.* 6(2), 4 (2007)
16. Altschul, S.F., Gish, W., Miller, W., Myers, E.W., Lipman, D.J.: Basic Local Alignment Search Tool. *J. Mol. Biol.* 215(3), 403–410 (1990)
17. Salton, G., Buckley, C.: Term-weighting approaches in automatic text retrieval. *Inf. Process Manag.* 24(5), 513–523 (1988)
18. Leitner, F., Krallinger, M., Rodriguez-Penagos, C., Hakenberg, J., Plake, C., Kuo, C.J., Hsu, C.N., Tsai, R.T., Hung, H.C., Lau, W.W., Johnson, C.A., Saetre, R., Yoshida, K., Chen, Y.H., Kim, S., Shin, S.Y., Zhang, B.T., Baumgartner Jr., W.A., Hunter, L., Haddow, B., Matthews, M., Wang, X., Ruch, P., Ehrler, F., Ozgür, A., Erkan, G., Radev, D.R., Krauthammer, M., Luong, T., Hoffmann, R., Sander, C., Valencia, A.: Introducing meta-services for biomedical information extraction. *Genome Biol.* 9(suppl. 2), S6 (2008)
19. Goble, C.A., Belhajjame, K., Tanoh, F., Bhagat, J., Wolstencroft, K., Stevens, R., Pettifer, S., Nzuobontane, E., McWilliam, H., Laurent, T., Lopez, R.: BioCatalogue: a curated Web Service registry for the Life Science community. In: ISMB/ECCB 2009. Technology Track: TT40 (2009)
20. Louie, B., Mork, P., Martin-Sanchez, F., Halevy, A., Tarczy-Hornoch, P.: Data integration and genomic medicine. *J. Biomed. Inform.* 40(1), 5–16 (2007)
21. Pihur, V., Datta, S., Datta, S.: Weighted rank aggregation of cluster validation measures: a Monte Carlo cross-entropy approach. *Bioinformatics* 23(13), 1607–1615 (2007)
22. DeConde, R., Hawley, S., Falcon, S., Clegg, N., Knudsen, B., Etzioni, R.: Combining results of microarray experiments: a rank aggregation approach. *Stat. Appl. Genet. Mol. Biol.* 5, Article 15 (2006)
23. Pihur, V., Datta, S., Datta, S.: RankAggreg, an R package for weighted rank aggregation. *BMC Bioinformatics* 10, 62 (2009)
24. Fagin, R., Kumar, R., Sivakumar, D.: Comparing top k lists. *SIAM J. Discrete Math.* 17(1), 134–160 (2003)
25. Börzsönyi, S., Kossmann, D., Stocker, K.: The Skyline operator. In: Proceedings 17th International Conference on Data Engineering, ICDE 2001, pp. 421–430. IEEE Press, New York (2001)
26. Hue, C., Boullé, M.: A new probabilistic approach in rank regression with optimal bayesian partitioning. *J. Mach. Learn. Res.* 8, 2727–2754 (2007)
27. Cheung, C.W.: Probabilistic rank aggregation for multiple SVM ranking. MPhil Thesis. Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, Hong Kong (2009)
28. Sawaragi, Y., Nakayama, H., Tanino, T.: Theory of multiobjective optimization. *Mathematics in Science and Engineering*, vol. 176. Academic Press Inc., Orlando (1985)
29. Steuer, R.E.: Multiple criteria optimization: theory, computations, and application. John Wiley & Sons, Inc., New York (1986)
30. Deb, K.: Multi-objective optimization using evolutionary algorithms. John Wiley & Sons, Inc., New York (2002)
31. Deb, K., Agrawal, S., Pratap, A., Meyarivan, T.: A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. KanGAL Report no. 200001 (2000)
32. Zitzler, E., Thiele, L.: An evolutionary algorithm for multiobjective optimization: the strength Pareto approach. TIK-Report no. 43 (1998)

33. Handl, F., Kell, D.B., Knowles, J.D.: Multiobjective optimization in bioinformatics and computational biology. *IEEE/ACM Trans. Comput. Biol. Bioinform.* 4(2), 279–292 (2007)
34. Perez-Iratxeta, C., Bork, P., Andrade, M.A.: Association of genes to genetically inherited diseases using data mining. *Nat. Genet.* 31(3), 316–319 (2002)
35. Jelier, R., Jenster, G., Dorssers, L.C., van der Eijk, C.C., van Mulligen, E.M., Mons, B., Kors, J.A.: Co-occurrence based meta-analysis of scientific texts: retrieving biological relationships between genes. *Bioinformatics* 21(9), 2049–2058 (2005)
36. Kerr, G., Ruskin, H.J., Crane, M., Doolan, P.: Techniques for clustering gene expression data. *Comput. Biol. Med.* 38(3), 283–293 (2008)
37. Kearsley, M.J.: The principles of QTL analysis (a minimal mathematics approach). *J. Exp. Bot.* 49(327), 1619–1623 (1998)
38. Datta, R., de Schoolmeester, M.L., Hedeler, C., Paton, N.W., Brass, A.M., Else, K.J.: Identification of novel genes in intestinal tissue that are regulated after infection with an intestinal nematode parasite. *Infect. Immun.* 73(7), 4025–4033 (2005)



# Appendix A:

## Search Computing Dictionary

### 1 Service Framework

#### Service Mart

Data abstraction modeling → **data sources** referring to the same kind of “Web object” (e.g., hotels, movies, etc.). Service marts are described at three different levels: a higher level describing the → **service mart signature** and → **connection patterns**, an intermediate level describing all the → **service interfaces** available for the → **service mart signature**, and a lower level describing the → **service implementation** of each **service interface**.

#### Data Source

A ◊ **data repository** that can be accessed programmatically, such as a ◊ **database**, an ◊ **application program**, a ◊ **Web Service**, a ◊ **wrapper** extracting data from a ◊ **Web site**, etc.

#### Service Mart Signature

A characterization of a → **service mart** in ◊ **relational** terms. It consists of the name of the service mart and the set of → **attributes of the service mart**.

#### Attributes of a Service Mart

Attributes characterizing the different ◊ **fields** that are handled by the → **service mart** to describe properties of a Web object. Each attribute is associated with a → **built-in type** and possibly with a set of → **keywords**. The set of attributes of a service mart can be either single-valued or multi-valued. Multi-valued attributes can be put together to form a → **repeating group of attributes**.

#### Repeating Group of Attributes

A set of multi-valued attributes of a → **service mart** that collectively define one ◊ **property** of a Web object. Within the same Web object, the attributes of the same repeating group all have the same number of values.

#### Built-in Type

Any concrete type of a programming language supporting Search Computing, such as int, string, float, double, date, ...

#### Keyword

A term characterizing the semantics of an attribute. Keywords are normally taken from vocabularies, e.g. ◊ **WordNet**.

#### Service Interface

A characterization of one of the possible → **service implementations** of a → **service mart** given by one ◊ **URI**, one set of → **Service Parameters**, and one → **access**

**pattern.** The same  $\rightarrow$  **access pattern** is shared by possibly more than one service interface. Every service interface has exactly one service implementation.

### Adornment

A pair consisting of an  $\rightarrow$  **access pattern** and a  $\rightarrow$  **scoring function** associated to a  $\rightarrow$  **service interface**. The same adornment is shared by possibly more than one service interface.

### Adorned Service Mart

The association of a  $\rightarrow$  **service mart** with an  $\rightarrow$  **adornment** existing in at least one  $\rightarrow$  **service interface** of the service mart.

### Service Implementation

A concrete  $\diamond$  **implementation** of one  $\rightarrow$  **service interface**, thus providing operations for accessing data available at one or more  $\rightarrow$  **data sources** according to the  $\rightarrow$  **access pattern** of the service interface.

### Access Pattern

An labeling of a  $\rightarrow$  **service mart signature** that determines which attributes of the  $\rightarrow$  **service mart** are  $\diamond$  **output** and which ones are  $\diamond$  **input** attributes. Among  $\diamond$  **output** attributes, some may be labeled as ranked and associated with a  $\rightarrow$  **score**, and in that case the service implementation returns its results in an order which depends on them. There can be several  $\rightarrow$  **service interfaces** for the same  $\rightarrow$  **access pattern**.

### Scoring Function

Function mapping each tuple output by a  $\rightarrow$  **service implementation** into a  $\rightarrow$  **score**. The mapping can be determined based on the  $\rightarrow$  **attributes of the service mart** of the  $\rightarrow$  **service interface** associated with the scoring function, or the position of the tuple in the output, and depends on the  $\rightarrow$  **ranking type**. Note that if the  $\rightarrow$  **ranking type** is unranked, then the scoring function is a fixed constant (e.g., 1).

### Connection Pattern

Specification of the join between two  $\rightarrow$  **service marts**, expressed as a conjunctive expression of join predicates using the  $\rightarrow$  **attributes of the service marts**. Every join predicate is built by a pair of  $\rightarrow$  **service mart attributes**, orderly taken from the first and second  $\rightarrow$  **service mart**, and an arbitrary  $\diamond$  **comparison operator**.

### Service Mart Design

Process of defining the signature of a  $\rightarrow$  **service mart**, then its  $\rightarrow$  **access patterns**, then the  $\diamond$  **data sources** that can support queries expressed with those  $\rightarrow$  **access patterns**, each of which is registered as a  $\rightarrow$  **service interface**. The process is completed by defining  $\rightarrow$  **connection patterns** between pairs of  $\rightarrow$  **service marts**.

### Service Mart Implementation

Production of a  $\rightarrow$  **service implementation** for a given  $\rightarrow$  **service interface**. The process may use components and tools for data materialization, extraction, conversion, and translation.

**Service Registration**

Addition of a  $\rightarrow$  **service implementation** and its  $\rightarrow$  **service interface** to a  $\rightarrow$  **service mart**. With the service registration, the service implementation is made available at the  $\diamond$  **URI** specified in the service interface.

**Ranking Type:**

A set of properties regarding the ranking, i.e., the order in which the results of consecutive  $\rightarrow$  **fetches** performed on a  $\rightarrow$  **service implementation** are returned. The ranking type defines:

- whether a  $\rightarrow$  **service implementation** is  $\diamond$  **ranked** or  $\diamond$  **unranked**,
- whether it is opaque or visible (i.e., the  $\rightarrow$  **attributes of the service mart** include a  $\rightarrow$  **score**),
- the set of output attributes the ranking depends on (the set is empty if it does not depend on output attributes)
- the  $\rightarrow$  **scoring function** mapping the set of output attributes the ranking depends on into a  $\rightarrow$  **score**, and
- whether the ranking is  $\diamond$  **ascending** or  $\diamond$  **descending**.

**Search Service**

Service implementation that returns  $\rightarrow$  **chunks** of ranked results that are possibly  $\diamond$  **unbounded** in number. The corresponding ranking type is  $\diamond$  **ranked**.

**Exact Service**

Service implementation whose corresponding ranking type is  $\diamond$  **unranked**. The results returned by such a service implementation are  $\diamond$  **finite** and not  $\rightarrow$  **chunked**, except for technical reasons.

**Service Parameter:**

Meta data describing the  $\rightarrow$  **service implementation** of a  $\rightarrow$  **service mart**, being one of  $\rightarrow$  **ranking type**,  $\rightarrow$  **cacheable**,  $\rightarrow$  **cache time-to-live**,  $\rightarrow$  **isChunked**,  $\rightarrow$  **chunk size**,  $\rightarrow$  **ERSPI**,  $\rightarrow$  **decay factor**,  $\rightarrow$  **response time**, and  $\rightarrow$  **cost**.

**Fetch**

Request for the next  $\rightarrow$  **chunk** of results from a  $\rightarrow$  **service implementation**.

**Chunked**

Property of a service implementation whose results are organized in  $\rightarrow$  **chunks**.

**Chunk**

Any page of results in the form of  $\diamond$  **tuples** returned by a  $\rightarrow$  **service implementation**.

**Cacheable**

Parameter of a  $\rightarrow$  **service interface**. True if the results returned by a  $\rightarrow$  **service implementation** can be stored in a  $\diamond$  **cache**; false otherwise.

**Cache Time-to-Live**

Parameter of a  $\rightarrow$  **service interface**. Duration of the validity of  $\diamond$  **cached data**.

**isChunked**

Parameter of a  $\rightarrow$  **service interface**. True if the  $\rightarrow$  **service implementation** is  $\rightarrow$  **chunked**; false otherwise.

**Chunk Size**

Parameter of a  $\rightarrow$  **service interface**. The number of  $\diamond$  **tuples** in a  $\rightarrow$  **chunk**.

**ERSPI (Expected Result Size Per Invocation)**

Parameter of a  $\rightarrow$  **service interface**.  $\diamond$  **Positive real number** expressing the expected size of the result produced by a  $\rightarrow$  **service implementation** when called with arbitrary input values; in  $\diamond$  relational terms, expresses the  $\diamond$  **average cardinality** of the result of a query. The ERSPI is not very significant with search services, as the complete result may consist of a very high number of  $\diamond$  **tuples**, which however are not completely retrieved by fetch operations.

**Selectivity**

$\diamond$  **Positive real number** expressing the ratio between  $\rightarrow$  **ERSPI** of a  $\rightarrow$  **service implementation** and the total number of tuples that can be returned by the  $\rightarrow$  **service implementation**.

**Decay Function**

Parameter of a  $\rightarrow$  **service interface**. A function that models the decay of the  $\rightarrow$  **scores** of the results returned by consecutive  $\rightarrow$  **fetches**.

**Response Time**

Parameter of a  $\rightarrow$  **service interface**. Average  $\diamond$  **response time** of a  $\rightarrow$  **fetch**.

**Cost**

Parameter of a  $\rightarrow$  **service interface**.  $\diamond$  **Monetary cost** of a  $\rightarrow$  **fetch**.

**Score**

Value in the [0,1] interval indicating the quality (1=highest, 0=lowest) of a  $\diamond$  **tuple** according to a predefined criterion as specified by a  $\rightarrow$  **scoring function**.

## 2 Query Expression and Optimization

**Query on Service Marts**

A query is a  $\diamond$  **graph** whose nodes are labeled with  $\rightarrow$  **service marts** and whose edges are labeled with  $\rightarrow$  **connection patterns**. Nodes are additionally labeled with  $\diamond$  **selection predicates** over the  $\rightarrow$  **service mart attributes** and  $\diamond$  **projected attributes** forming the result. The same  $\rightarrow$  **service mart** can appear multiple times in the same

query. Every query is interpreted as a  $\diamond$  **conjunctive query**, whose  $\diamond$  **join predicates** are built from  $\rightarrow$ **connection patterns**.

### Query on Adorned Service Marts

A query is a  $\diamond$  **graph** whose nodes are labeled with  $\rightarrow$  **adorned service marts** and whose edges are labeled with  $\rightarrow$  **connection patterns**. Additional labeling is as for  $\rightarrow$ **queries on service marts**. A  $\rightarrow$ **query on service marts** can be turned into a query on adorned service marts simply by substituting to every  $\rightarrow$ **service mart** labeling a query node one of the corresponding  $\rightarrow$  **adorned service marts**.

### Feasible Query

A  $\rightarrow$ **query on adorned service marts** is feasible if the  $\rightarrow$ **access patterns** used in the query satisfy given well-formedness conditions guaranteeing that it is possible to compute the results as if access patterns were not present.

### Query on Service Interfaces

A  $\rightarrow$ **query on adorned service marts** can be turned into a query on service interfaces simply by substituting to every  $\rightarrow$  **adorned service mart** labeling a query node one of its  $\rightarrow$ **service interfaces**. Recall that every  $\rightarrow$  **service interface** is also associated with a  $\rightarrow$  **service implementation**.

### Query Formulation

The process of specifying  $\rightarrow$  **feasible queries** on  $\rightarrow$  **service interfaces** and possibly a  $\rightarrow$  **ranking function** associated with the query. Suitable interfaces assist users in formulating only  $\rightarrow$ **feasible queries**.

### Query Processing

The process of executing  $\rightarrow$  **feasible queries** on  $\rightarrow$  **service interfaces**, producing a  $\rightarrow$  **query result**. The process consists of (1) query installation (2) query optimization (3) query execution (4) user interaction for interpreting results, possibly leading to further steps of query processing.

### Query Result

A  $\diamond$  **list** of  $\rightarrow$  **combinations**.

### Combination

A  $\diamond$  **tuple** of the  $\rightarrow$  **query result**, built by the matching  $\diamond$ **tuples** produced during  $\rightarrow$  **query execution**, where a query result  $\diamond$  **tuple** includes exactly one  $\diamond$  **tuple** from the results produced by each  $\rightarrow$  **service implementations** involved in the query execution. Combinations in the query result are ordered according to a  $\rightarrow$  **ranking function** associated with the query.

### Ranking Function

A  $\diamond$  **weighted sum** of the individual  $\rightarrow$  **scores** of the  $\diamond$  **tuples** forming a  $\rightarrow$  **combination**, which returns the ranking of the  $\rightarrow$  **combination** according to the users' preferences.

### Query Optimization

The process of building query plans for  $\rightarrow$  **correct queries**, and then selecting the  $\rightarrow$  **optimal query plan** which is then executed.

### Query Plan

A  $\diamond$  **directed acyclic graph** made of  $\rightarrow$  **nodes of a query plan** and  $\rightarrow$  **edges of a query plan**, representing an operational specification of the order in which  $\rightarrow$  **service interfaces** are to be invoked in order to retrieve a specified number of  $\rightarrow$  **chunks**, and of how  $\rightarrow$  **joins** are to be performed between  $\rightarrow$  **chunks** according to specific  $\rightarrow$  **join strategies**.

### Node of a Query Plan

Any node in a  $\rightarrow$  **query plan**, representing the  $\rightarrow$  **initial node**, or the  $\rightarrow$  **final node**, or an  $\rightarrow$  **invocation node**, or a  $\rightarrow$  **selection node**, or a  $\rightarrow$  **join node**.

### Edge of a Query Plan

A directed edge in a  $\rightarrow$  **query plan**, representing the data transfer between two  $\rightarrow$  **nodes of a query plan**

### Initial Node

A  $\rightarrow$  **node of a query plan** with no incoming  $\rightarrow$  **edges** and one or more outgoing  $\rightarrow$  **edges**, used to denote the user input. It is the only node in the  $\rightarrow$  **query plan** without incoming edges, and every  $\rightarrow$  **query plan** must have exactly one initial node.

### Final Node

A  $\rightarrow$  **node of a query plan** with no outgoing  $\rightarrow$  **edges** and one or more incoming  $\rightarrow$  **edges**, used to denote the production of the query result. It is the only node in the  $\rightarrow$  **query plan** without outgoing edges, and every  $\rightarrow$  **query plan** must have exactly one final node.

### Invocation Node

A  $\rightarrow$  **node of a query plan** that represents the invocation of a  $\rightarrow$  **service interface**.

### Selection Node

A  $\rightarrow$  **node of a query plan** associated with a  $\diamond$  **selection operation**.

### Join

An operation between two  $\rightarrow$  **service interfaces**, called join operands, that uses  $\rightarrow$  **connection patterns** in order to form  $\rightarrow$  **combinations** of the  $\diamond$  **tuples** returned as results of service invocations. Joins are classified as  $\rightarrow$  **pipe joins** or as  $\rightarrow$  **parallel joins**, depending on the relationships between the  $\rightarrow$  **access patterns** associated with the join operands.

### Pipe Join

A configuration of two  $\rightarrow$  **invocation nodes** A and B in the  $\rightarrow$  **query plan** such that A and B are connected by a directed path from A to B, and one or more output

attributes of A contain values which are used as input in B (i.e., attributes are labeled as input in the  $\rightarrow$  **access pattern** of B).

### Pipe Node

The destination node in the directed path of a  $\rightarrow$  **pipe join** configuration.

### Parallel Join

A configuration of one  $\rightarrow$  **join node** and two antecedent nodes A and B in the  $\rightarrow$  **query plan** such that A and B correspond to service interfaces and neither the output attributes of A contain values that are used as input in B nor the output attributes of B contain values that are used as input in A.

### Join Node

A  $\rightarrow$  **node of a query plan** with exactly two incoming  $\rightarrow$  **edges of a query plan** and one or more outgoing  $\rightarrow$  **edges of a query plan**, used to denote a  $\rightarrow$  **parallel join**.

### Join Strategy

Defines the order in which  $\rightarrow$  **chunks**, received at a join or pipe node, are to be considered in order to produce  $\rightarrow$  **combinations** (i.e. join results). Different join strategies correspond to different orders in which the points in the  $\rightarrow$  **join search space** are to be considered. Join execution strategies can use the  $\rightarrow$  **nested loop** or  $\rightarrow$  **merge scan** strategies and perform a  $\rightarrow$  **rectangular exploration** or a  $\rightarrow$  **triangular exploration**.

### Nested Loop

A  $\rightarrow$  **join strategy** in which all  $\rightarrow$  **chunks** from one service are retrieved (and possibly cached) before proceeding to retrieve the next  $\rightarrow$  **chunk** from the other service.

### Merge Scan

A  $\rightarrow$  **join strategy** in which  $\rightarrow$  **chunks** from the two services being joined are alternatively retrieved in a fixed alternated order.

### Rectangular Exploration

An exploration of join combinations in which all available candidate combinations are considered as soon as new chunks are made available. The rectangular strategy can be applied to both  $\rightarrow$  **merge scan** and  $\rightarrow$  **nested loop**.

### Triangular Exploration

An exploration of join combinations in which only the one half of the available candidate yielding to better rankings are considered as soon as new chunks are made available. The triangular strategy can be applied to both  $\rightarrow$  **merge scan** and  $\rightarrow$  **nested loop**.

### Cost Model

A selection of the relevant  $\rightarrow$  **service parameters** and mathematical relationships thereof for assessing the execution cost of a query, representing a specific objective to guide the optimization process.

**Cost Function**

A mathematical function of some chosen  $\rightarrow$  **service parameters** whose result is a measure of the  $\rightarrow$  **cost of a query plan**.

**Cost of a Query Plan**

The result of applying a specific  $\rightarrow$  **cost function** with a specific  $\rightarrow$  **cost model** to a specific  $\rightarrow$  **query plan**.

**Optimal Query Plan**

The  $\rightarrow$  **query plan** whose  $\rightarrow$  **cost** is minimal among all possible plans for a given query.

### 3 Query Execution

**Execution Plan**

Directed graph consisting of nodes in the form of  $\rightarrow$  **scheduler units** and edges in the form of either  $\rightarrow$  **data flow edges** and  $\rightarrow$  **control flow edges**. An execution plan represents the physical evaluation of a  $\rightarrow$  **query plan**.

**Scheduler Unit**

A node of the  $\rightarrow$  **execution plan** that has a well defined semantics in terms of an operation that it performs within the execution plan.  $\rightarrow$  **producer Unit** and  $\rightarrow$  **consumer Unit**.

**Producer Unit**

A scheduler unit that produces output data, i.e. a publisher.  $\rightarrow$  **service Invocation unit**,  $\rightarrow$  **(parallel) join unit**,  $\rightarrow$  **ranker unit**,  $\rightarrow$  **chunker unit**,  $\rightarrow$  **selection unit**, and  $\rightarrow$  **cache unit**.

**Consumer Unit**

A scheduler unit that consumes input data, i.e. a subscriber.  $\rightarrow$  **service invocation unit**,  $\rightarrow$  **(parallel) join unit**,  $\rightarrow$  **ranker unit**,  $\rightarrow$  **chunker unit**,  $\rightarrow$  **selection unit**, and  $\rightarrow$  **cache unit**.

**Control Flow Edge**

An edge of the execution plan that transmits a control signal from one unit to another. Signals are of three kinds:  $\rightarrow$  **pulse signals**,  $\rightarrow$  **suspend / resume signals**.

**Data Flow Edge**

An edge of the execution plan that transports data,  $\rightarrow$  **chunks** of tuples, from a  $\rightarrow$  **producer unit** to a  $\rightarrow$  **consumer unit**.

**Clock Unit**

Controls  $n > 0$   $\rightarrow$  **service invocation units** using  $\rightarrow$  **pulse signals** which are produced at each  $\rightarrow$  **clock cycle**. A clock unit is controlled by a  $\rightarrow$  **clock function** and adjusted by  $\rightarrow$  **clock modifiers**.



### Pulse Signal

Pulse signals are emitted by the clock and trigger → **service invocation units** to fetch data. They contain the specification of the number of → **fetches** to be performed in each → **clock cycle**. A pulse signal can as well be produced by a data → **producer unit** when it first produces a tuple in output.

### Clock Frequency

Parameter of the → **clock unit** that describes how many → **clock cycles** per time unit a clock has. Example: 1/50 s means 50 clock cycles per second.

### Clock Cycle

The period during which the clock triggers the services according to current  $n$ -tuple of the → **clock function**. The number of clock cycles per time unit is determined by the → **clock frequency**.

### Clock Function

A clock function is a regular expression that describes, for each → **service invocation unit** controlled by the clock and for each → **clock cycle**, the number of → **fetches** to be performed by the Service Invocation Unit. A regular expression for a clock function maps into a sequence of clock values given as  $n$ -tuples of  $\diamond$  **integer numbers**, where  $n$  is the number of Service Invocation Units controlled by the → **clock unit**. As an example,  $(1,1)(2,2)^*$  represents a sequence in which two services are invoked once in the first clock cycle, and twice in any subsequent clock cycle. Each clock cycle of a clock function is expected to control the → **join execution strategy** in terms of → **chunks** produced.

### Clock Modifier

A clock modifier for a → **clock unit** is an  $\diamond$   $n$ -tuple of  $\diamond$  **positive real numbers**, where  $n$  is the number of → **service invocation units** controlled by that clock unit. During query execution, clock modifiers are used to adjust the number of → **fetches** for each Service Invocation Unit controlled by the clock unit.

### Suspend/Resume Signals

A suspend signal is sent by the → **parallel join unit** to the → **clock unit** once the → **skew factor** of the join unit has been reached; it is also sent when a given number  $k$  of results have been produced. On receipt of a suspend signal, the clock suspends execution only at the end of a clock cycle. As soon as the → **join execution strategy** realigns, a resume signal is sent to the clock to continue query execution. Suspend and resume signals can as well be produced by → **end users**, the former occur when they want to suspend the execution of an execution plan, e.g. because they are temporarily satisfied with the current results, the latter when they want to resume execution of an → **execution plan**.

### Chunker Unit

A → **scheduler unit** that, based on the specification of a → **chunking function**, constructs new → **chunks** with the tuples it gets as input. Internally, the chunker unit breaks up the chunks in input and produces new chunks in output.

### Chunking Function

A specification, given in the form of a regular expression, of how many tuples of the input are to be combined into every chunk in output. Example:  $4^{\wedge}3, 3^*$  means: first produce 3 chunks with 4 tuples, then continue with chunks of 3 tuples.

### Service Invocation Unit

A service invocation unit fetches data by using a specific  $\rightarrow$  **access pattern** of a given  $\rightarrow$  **service mart**. Service invocation units can either invoke  $\rightarrow$  **search services** or  $\rightarrow$  **exact services**. Service invocation units are triggered by a  $\rightarrow$  **pulse signal** or by the availability of input data. When a service invocation follows one or more service invocations in the execution plan, it implicitly computes a  $\rightarrow$  **pipe join**.

### Parallel Join Unit

It joins the results of two search service calls; executing a join causes the production of result tuples, called  $\rightarrow$  **combinations**, according to a  $\rightarrow$  **join execution strategy**. Joins are associated with a  $\rightarrow$  **Skew** value and can emit  $\rightarrow$  **suspend/resume Signals** to the clock that controls the  $\rightarrow$  **service invocation units** generating the data for the join unit.

### Skew

In the case of runtime service behavior diverging from the expected one,  $\rightarrow$  **parallel join units** are allowed to deviate from their predefined  $\rightarrow$  **join execution strategy**. The maximum permitted deviation is specified by the skew value. It states how many  $\rightarrow$  **chunks** on the x or y axis can be processed, in addition to the planned ones, before the join unit sends a  $\rightarrow$  **pause signal** to its clock. The signal stops the production of new chunks and permits the join unit to wait for already fetched chunks to propagate through the  $\rightarrow$  **query execution plan**.

### Cache Unit

Cache units store the output generated by  $\rightarrow$  **producer units** and make it available to  $\rightarrow$  **Consumer Units**. Therefore they serve as a common point of synchronization in the producer/consumer (or publish/subscribe) model. Inserts of data that is already cached have no effect. Caches may trigger their consumers whenever new data is available.

### Ranker Unit

Ranks or re-ranks tuples in  $\rightarrow$  **chunks** according to a  $\rightarrow$  **ranking function**. It has a blocking behavior, which is controlled by the specification of the ranking function. More precisely, the unit re-ranks tuples up to a certain size or time limit, after which it outputs the ranked tuples and resets itself. It may therefore work as a synchronization point when receiving tuples from more than one unit.

### Selection Unit

Performs selections (consisting of  $\diamond$  **Boolean expressions** of  $\diamond$  **selection predicates**) over the dataflow  $\diamond$  **tuples** in input; only  $\diamond$  **tuples** satisfying the selection are produced in output.

## 4 Liquid Queries

### Resource Graph

Graph showing to → **expert users** nodes labeled with → **service marts** or → **service interfaces** and arcs labeled with → **connection patterns**; → **expert users** build queries by means of graphic interfaces, with a  $\diamond$  **mash-up** style. Query formulation tools include facilities for selecting nodes and arcs, selecting result attributes, describing the rank function, and describing possible query augmentations.

### Expert User

Knows about resources and their semantics, assembles queries and defines → **query interfaces** for personal use but also for saving them and making them available to → **end users**.

### End User

Interacts with query by submitting values into query forms, possibly within slots which are well-defined in terms of semantics and with expected data types (for matching with given → **attributes of service marts**).

### Liquid Query

Query whose formulation is flexible and fluid, with the purpose of discovering the user intent while submitting queries and reading results, in a more precise manner, through a step-by-step approach that allows the user to get closer and closer to the desired information. A liquid query consists of an initial → **query interface**, may support several → **query augmentations**, and produces a → **liquid result**.

### Query Expansion

Possibility of adding to a query a simple sub-query, consisting of joining one of the → **service interfaces** already in the query to another → **service interface**, connected to it though a → **connection pattern**. The process is recursive and can be applied to the → **result page** or to a subset of the → **result combinations** shown in the → **result page**.

### Liquid Result

a set of results of a liquid query, which in turn is flexible thanks to a set of → **result interaction primitives** that allow the end-user to modify the → **result structure** and set of → **result instances**, possibly shown partitioned in → **result pages**.

### Query Interface

A  $\diamond$  **user interface** that is offered for formulating the initial query. It consists of a set of → **search service forms**, which in turn are composed by → **search fields**.

### Result Instance

Result tuple of a → **liquid query**, that complies with the corresponding → **result structure**.

**Result Page**

Set of → **result instances** that are shown together within the → **liquid result** interface.

**Result Structure**

The ◇ **schema** of the result of a liquid query. It consists of a set of typed attributes. See also → **Attributes of a Service Mart**.

**Search Fields**

Input fields that compose a → **search service form**.

**Search Service Form**

Web form that allows user to submit parameters required by a search service in a → **query interface**.

**Result Interaction Primitive**

User command enabled within → **liquid results** to refine, modify, extend the result itself (either in terms of → **result structure** or → **result instances**). Result interaction primitives are listed at the end of this dictionary and are marked with the symbol §.

**§ Group**

Collecting in a group the results having common values for a specified attribute. The group assumes as a title the attribute value at hand. By applying this operation, all the clustering and sorting options possibly defined by the user are applied separately to each group.

**§ Ungroup**

Removing the existing grouping.

**§ Cluster**

Clustering adjacent tuples on a specific attribute and hiding duplicate valuesDefining a new grouping on the results.

**§ Uncluster**

Removing a clustering on the results.

**§ Sort**

Sorting the results currently being displayed according to one or more different attributes with respect to the ones initially defined. Sorting can be ascending or descending.

**§ Unsort**

Removing a sorting on the results.

**§ Drill-Down**

Visualization of (already available but currently hidden) additional attributes.

**§ Roll-Up**

Hiding some attributes that are currently visible (and accordingly removing duplicates).

**§ Filter**

Reducing the number of shown results based on a simple condition on one attribute.

**§ DeleteInstance**

Locally deleting one instance from the currently displayed items. This can be seen as a particular case of filter operation.

**§ RemoveFilter**

Canceling a filter currently applied.

**§ More (all)**

Asking for more results from the same query, by extracting results from every involved → **service mart**.

**§ More of One Service**

Asking for more results on a limited set of → **service marts** involved in the query.

**§ Expand**

Asking for additional results (coming from other → **service marts**, connected through predefined → **connection patterns**) on a limited set of items listed in the current result set of the query.

**§ Change Ranking**

Ranking the results of the query according to a different ranking function wrt the one initially defined. This might produce a different set of results than those displayed before re-ranking.

**§ Pivoting**

Clustering together all instances with the same multivalued attribute value or repeating group value, thereby rendering the other attributes as repeating groups.

**§ ChangeProvider**

Changing the provider of a specific → **service interface**.

# Author Index

- Baeza-Yates, Ricardo 11  
Baumgartner, Robert 94  
Belhajjame, Khalid 114  
Bozzon, Alessandro 135, 244, 268  
Braga, Daniele 188, 225  
Brambilla, Marco 244, 268  
Buganza, Tommaso 45  
  
Campi, Alessandro 94, 114, 163  
Casati, Fabio 72  
Ceri, Stefano 3, 163, 188, 225, 244, 268  
Corcoglioniti, Francesco 225, 268  
  
Daniel, Florian 72  
Della Valle, Emanuele 45  
  
Embury, Suzanne M. 114  
  
Fernandes, Alvaro A.A. 114  
Fraternali, Piero 135, 244  
  
Gatti, Nicola 268  
Gottlob, Georg 94, 163  
Grossniklaus, Michael 188, 225  
  
Hedeler, Cornelia 114  
Herzog, Marcus 94  
  
Ilyas, Ihab F. 211  
  
Maesani, Andrea 163  
Manolescu, Ioana 244  
Martinenghi, Davide 211  
Masseroli, Marco 291  
  
Paton, Norman W. 114, 291  
  
Raghavan, Prabhakar 11  
Ronchi, Stefania 163  
  
Soi, Stefano 72  
Spasić, Irena 291  
  
Tagliasacchi, Marco 211  
  
Weikum, Gerhard 24